DOI: 10.24275/uama.6732.8905

Universidad Autónoma Metropolitana
Casa abierta al tiempo Azcapotzalco

Comparativa de métodos para la optimización de parámetros en el algoritmo QAOA

Idónea comunicación de resultados

Presenta:
Ing. Brian García Sarmina

Asesores:
Dr. Salas Brito Álvaro Lorenzo
Dr. Mora Gutierrez Román Anselmo

Idónea comunicación de resultados presentada para obtener el título de la Maestría en Ciencias de la Computación

Unidad Azcapotzalco
Universidad Autónoma Metropolitana
Ciudad de México, México.

18 de enero del 2022

Agradecimientos

Quiero empezar por agradecer al amor de mi vida. Mariana, todos los días agradezco que seamos pareja, que seamos amigos y sobre todo compañeros, estos últimos años han sido muy duros pero de éxitos también. Nuestra historia está llena de lecciones, besos, abrazos, risas y momentos únicos, de metas que hemos alcanzado y metas que aún nos faltan por cruzar, pero al final solo tenemos que recordar que esta vida hay que vivirla lo mejor que podamos, siendo felices y haciendo lo que amamos, para que el día que nos toque contarla, la contemos juntos tomados de la mano, recordando todo nuestro camino con una sonrisa... Te amo.

Mamá, quiero agradecerte y dedicarte este nuevo logro de mi vida, se cuanto te has esforzado por apoyarme y ayudarme a conquistar mis sueños, se que a veces la vida puede ser muy difícil pero tú nunca pierdes tu sonrisa ni tu fe en mi. Esto solo es la mitad del camino, verás que llegaré hasta donde les prometí a ti y a mi papá que llegaría.

Hermanos, Jonathan y Gibran, también les agradezco a ustedes por todo lo que han hecho para ayudarme a completar este nuevo paso, les doy las gracias por su apoyo en todos los aspectos, y quiero que sepan que aunque no se los digo con palabras, se los digo con mis acciones, y esta es una de ellas, una meta más que comparten ustedes conmigo.

Quiero agradecerle también a mi sobrino José Miguel, porque tú tal vez aún no te das cuenta, pero eres una persona muy importante en mi vida. Agradezco que siempre tengas una sonrisa, un abrazo, y un momento de inocencia que detienen mi vida para darme cuenta que a veces los problemas no son tan complejos como uno los piensa, y si algo no sale bien, sé que tú me dirás: “tengo un pian”.

Este trabajo se los dedico a todos mis seres queridos, los que están con nosotros y los que ya no lo están. Aún faltan muchos capítulos por escribir, y cada uno será mejor. Gracias por todo.

Resumen

El siguiente trabajo se aborda el estudio comparativo de dos métodos de optimización dentro del Quantum Approximation Optimization Algorithm (QAOA), estos métodos de optimización son la Búsqueda Exhaustiva y la heurística de Búsqueda Local Iterada o Iterativa.

Ambos métodos son probados en varios problemas, estos problemas se separan en dos categorías, la primera categoría son los problemas de Ising Spin Model (ISM) y la segunda categoría son los problemas de Max-Cut. Dentro de cada categoría se tienen tres problemas distintos, cada uno difiere en el tipo de configuración que tiene, estas configuraciones son de tipo: lineal, cíclico y completo.

Además, la comparativa también se extiende a experimentaciones utilizando computadoras clásicas (simulaciones locales) y computadoras cuánticas (simulaciones reales), estas experimentaciones locales y reales permiten analizar la aplicabilidad de estos métodos de optimización dado el estado actual del hardware cuántico.

Índice general

Capítulo 1

Introducción

La primera mención sobre computadoras basadas en los principios de la mecánica cuántica puede ser asociada a Richard Feynman (1982), esta mención se relaciona con la capacidad de simular sistemas cuánticos en computadoras clásicas, ya que estos procesos contienen una gran cantidad de "estados posibles", que de alguna forma son similares a los problemas combinatorios clásicos, que al aumentar el número de partículas que interactúan en el proceso cuántico hacen que sea imposible realizar los cálculos necesarios para la simulación en una computadora clásica. Por otra parte, las computadoras cuánticas (basadas en los principios de la mecánica cuántica para realizar sus operaciones) en teoría no deberían de mostrar esa limitante, ya que tanto el sistema a simular como el dispositivo operan bajo los mismos principios físicos. [1] [3]

Esta mención de Richard Feynman se extendió a distintos ámbitos donde comenzaron a realizar grandes avances tomando en cuenta este nuevo paradigma de la computación, llegando con esto los famosos algoritmos de Transformada Rápida de Fourier, Shor, Grover, etc. Estos algoritmos han demostrado el gran potencial de la computación cuántica en distintas áreas de las ciencias de datos, en donde en los últimos años a existido un énfasis en poder utilizar este paradigma en problemas de Inteligencia Artificial y Big Data.

Esta tendencia se debe a que existen bastantes problemas donde se tienen una gran cantidad de resultados posibles que deban ser explorador para encontrar la respuesta óptima, también pueden tratarse de problemas que presenten un comportamiento de tipo no-convexo, o problemas que requieran de un alto grado de paralelización que se vuelven bastante complejos de tratar en escalas aplicables a problemas reales, y estas son solo algunas de las virtudes del cómputo cuántico con relación a problemas de optimización, aprendizaje máquina, y en ciencia de datos en general.

En el estado de investigación a la fecha sobre cómputo cuántico se conocen tres enfoques en los cuales una computadora cuántica tiene la capacidad de ser más eficiente que una clásica (incluso desde su desarrollo algorítmico), el primer tipo de problemas que se muestran con mejoría son los que respecta al uso de la **Transformada de Cuántica de Fourier** (usando el enfoque de Peter Shor)

que se usan principalmente para el problema de *factorización* y *logaritmo discreto*, la segunda clase de problemas son los basados en el **algoritmo de Grover** (paralelismo cuántico) para realizar procesos de *búsqueda de manera cuántica*, y por último, se tienen los problemas de simulación cuántica, esto deriva directamente de la suposición de Feynman donde se establece que un sistema cuántico solo se puede simular de forma eficiente en una computadora que sea capaz de usar estos mismos principios para operar. [8]

Actualmente, se estudian múltiples aplicaciones para algoritmos cuánticos, estos algoritmos usan distintas vertientes y métodos para resolver problemas a escala. Los algoritmos actuales buscan poder ser implementados en dispositivos NISQ "Noisy Intermediate Scale Quantum" que se encuentran en desarrollo y disponibles para el uso académico y de investigación por parte de algunas compañías que prestan su acceso de forma gratuita. Dentro de estos algoritmos se encuentra **QAOA** "Quantum Approximate Optimization Algorithm" que es un algoritmo que ha crecido bastante en popularidad en los últimos años (creado en 2014 [21]) esto se debe a que se ha descubierto que es bastante resistente a factores de ruido (por su modelo simple que necesita pocas compuertas para representar el problema) y que no necesita de qubits extra para ejecutar el algoritmo. Debido a estas características, QAOA es un algoritmo bastante interesante para estudiar de forma detenida para entender sus alcances, su potencial en dispositivos NISQ y su aplicabilidad en problemas reales a escala. [14] - [21]

En este trabajo se investiga respecto a los alcances y características que ofrece QAOA, con la finalidad de conocer su aplicabilidad para problemas actuales (a escala) y las limitaciones que este algoritmo tiene actualmente. Los experimentos que se realizarán para QAOA serán aplicando distintos métodos de optimización (para la búsqueda de los parámetros de QAOA) y se compararán estos métodos a través de simulaciones locales (en una computadora clásica) y simulaciones reales (realizadas en una computadora cuántica tipo NISQ) para identificar ventajas y desventajas de los métodos, y además conocer el estado actual de las computadoras NISQ con respecto a los resultados "ideales" de las simulaciones locales.

1.1. Objetivo general

Establecer una comparativa de distintos métodos de calibración para los parámetros de los operadores de fase y mezclador del algoritmo QAOA.

1.2. Objetivos específicos

- Analizar al menos dos métodos distintos para obtener los parámetros de los operadores de fase y mezclador de QAOA.

- Extraer las ventajas y desventajas de los distintos métodos de calibración y generar un análisis comparativo.
- Realizar simulaciones de manera local y simulaciones usando el hardware cuántico disponible de IBMQ y comparar los resultados obtenidos.
- Demostrar usando dos clases de problemas (a escala) la utilidad del algoritmo QAOA, así como sus virtudes y defectos en el estado actual del desarrollo de algoritmos cuánticos.

1.3. Planteamiento del problema

De manera general, el algoritmo QAOA (Quantum Approximation Optimization Algorithm) se compone de dos etapas, la primera etapa se focaliza en traducir y realizar la evaluación del problema usando los operadores de fase y mezclador con la finalidad de crear interferencia constructiva o destructiva que permiten que un estado aparezca o no con más o menos frecuencia, esta interferencia debe generar que el estado objetivo (esto aproximado óptimo del sistema) se presente con mayor frecuencia.

La segunda etapa consiste en "variar" o "calibrar" los parámetros internos de los operadores de fase y mezclador de tal forma que en cada iteración de prueba, el estado objetivo vaya incrementando su probabilidad de aparición. Aquí es cuando se presenta nuestro aporte, ya que actualmente se buscan distintos métodos para realizar la calibración de los parámetros, estas formas de optimización se muestran con ventajas, pero también con desventajas, por ello es importante explorar métodos que aún no han sido documentados que permitan obtener estos parámetros de manera eficiente, y con ello aportar al conocimiento general del comportamiento y alcances del algoritmo QAOA.

1.4. Breve historia de la computación cuántica

En 1980 Paul Bennioff, demostró las bases teóricas para realizar la construcción de una computadora cuántica basada en el modelo Hamiltoniano para describir la evolución de los procesos computables.

"That is, the whole computation process is described by a pure state evolving under the action of a given Hamiltonian. Thus all the component parts of the Turing machine are described by states which have a definite phase relation to one another as the calculation progresses... The existence of such models at least suggests that the possibility of actually constructing such coherent machines should be examined." [37]

En cualquier área de las ciencias computacionales (incluyendo la computación cuántica) es importante analizar el desempeño de los algoritmos, la notación más común para ello es la notación de la "gran-O" (big-O), esta notación analiza el límite superior del peor escenario posible, además, esta notación se relaciona con el orden del algoritmo, otra notación importante (y bastante común) es la notación de la "gran-Omega" (big-Omega) donde se trata el límite inferior del peor escenario posible.

Para el caso del algoritmo Deutsch-Jozsas, el algoritmo se dice que posee un "oráculo" que determina si la función f (con algún número de valores de entrada) es una constante o función balanceada. Este algoritmo en computación clásica se asocia con una complejidad de $O(n)$ pasos para resolverse, pero en una computadora cuántica el problema puede ser resuelto en "un solo paso", esto se logra por las diferentes características fundamentales de la computación cuántica, entre ellas, el paralelismo cuántico el cual permite (en este problema) generar un espacio donde se "evalúan" de manera simultánea todos los resultados para la función para determinar si esta es balanceada o no.

Bernstein-Vazirani (BV) investigaron el algoritmo de Deutsch-Jozsas, donde lograron probar una ventaja cuántica no-determinista del algoritmo, suponiendo la existencia de pequeños errores, con una complejidad de $O(n)$ para la computadora clásica, y $O(1)$ para la computadora cuántica. Además, Bernstein-Vazirani trabajaron en una versión cuántica de un algoritmo para calcular la transformada de Fourier llamado "QFT" (Quantum Fourier Transform).

Seth Lloyd, trabajó en el laboratorio de "Los Alamos" donde establecería el primer enfoque práctico para trabajar en una computadora cuántica (QC), él mencionó que los sistemas cuánticos permiten realizar nuevos tipos de operaciones, estas operaciones computacionales pueden usar una unidad básica (unidad cuántica) física representada como un punto cuántico (quantum dot), un spin nuclear, etc. Y si estas unidades cuánticas interactúan unas con otras podrían realizar cálculos de procesos cuánticos. Lloyd también mencionó las importantes propiedades que estas unidades cuánticas deben poseer, la más importante de ellas es que pueden encontrarse en superposiciones de diferentes estados.

Después de estos primeros pasos siguieron descubrimientos de algoritmos que cambiaron la perspectiva de los alcances de la computación cuántica, en específico estos algoritmos fueron los de Grover (para búsqueda no ordenada cuántica) y Shor (para factorización), estos dos algoritmos son los que comenzaron la "fiebre" por la computación cuántica, ya que los problemas que estos algoritmos podían resolver eran bastante difíciles de atacar de forma eficiente en computación clásica.

Derivado de los algoritmos de Grover y Shor (y entre otros avances), la investigación de la computación cuántica siguió desarrollándose, pero de forma más lenta, esto cambió a raíz de la creación de las primeras computadoras cuánticas, en las cuales se pudo probar ciertos aspectos de la teoría de Cómputo Cuánti-

co de manera práctica. Con la llegada de las primeras computadoras cuánticas, el avance en todo lo relacionado con Cómputo Cuántico comenzó a crecer considerablemente, creando nuevas áreas como la Informática Cuántica, Inteligencia Artificial Cuántica, etc. Actualmente, usando los dispositivos NISQ se han podido probar muchos algoritmos cuánticos para distintos problemas a escala, siendo el algoritmo QAOA uno de los más importantes, ya que ataca problemas de optimización combinatoria que pueden encontrarse en muchos problemas de computación clásica, o inclusive en problemas de otras áreas de la ciencia.

1.5. Estado del arte

El algoritmo QAOA es un algoritmo de optimización cuántico-clásico, además, debido a su funcionamiento es un algoritmo bastante importante y con gran expectativa, con un potencial de mejora en tiempo y almacenamiento computacional para problemas de optimización combinatoria, además, QAOA tiene la característica de poder ser implementado y probado en computadoras cuánticas actuales debido a su baja demanda de qubits y su poca profundidad (número de operaciones técnicas necesarias para llegar al resultado). Desde su creación en 2014, el algoritmo de QAOA se ha refinado en varios aspectos y probado en varios problemas de optimización, demostrando su potencial en problemas combinatorios. [10][13] - [16]

De manera general, el algoritmo QAOA funciona en dos secciones, la primera sección genera un método de combinatoria basada en la alternancia de las rotaciones de los estados de los qubits (operadores de fase), a dichos qubits se les aplica una función de penalización (operadores mezcladores) que generan interferencia constructiva o destructiva. [11] La segunda sección del algoritmo se basa en la búsqueda de mejores valores para los parámetros de los operadores de fase y mezcladores, esta etapa en el enfoque tradicional es realizada por un algoritmo de optimización clásico, como lo puede ser el algoritmo del descenso del gradiente, u otro método de optimización clásico. [10]

El algoritmo de QAOA se puede considerar como un algoritmo híbrido cuántico-clásico, y actualmente existen métodos "zeroth-order" que permiten realizar aproximaciones directas al gradiente utilizando una función de pérdida, también existen algunas estrategias que utilizan distintos algoritmos cuánticos para evaluar el gradiente, sin embargo, estas técnicas usualmente requieren de un gran número de mediciones, y por consiguiente, una gran cantidad de repeticiones del circuito cuántico para poder llegar a un buen valor del gradiente descendiente. [10] [12]

Dado que a la fecha se conocen muy pocas implementaciones del algoritmo QAOA que realice sus dos secciones generales de operación usando solo algoritmos cuánticos, aún no se puede establecer la superioridad de un método de ejecución completamente cuántico sobre las versiones híbridas cuántico-clásico para QAOA.

[14] - [21] [25] [26]

1.5.1. Enfoques clásicos para optimizar parámetros

Otro enfoque aplicable en QAOA corresponde a los algoritmos "clásicos" los cuales buscan implementar nuevas estrategias de optimización clásica para realizar las tareas de cálculo para los parámetros de los operadores de fase y mezcladores para el algoritmo QAOA.

El primer método investigado fue el propuesto por *Koch, D. et al. (2020)* quienes presentan un texto de los algoritmos cuánticos más populares (actualmente) y algunos ejemplos de aplicación de los mismos. En el trabajo de *Koch, D. et al. (2020)* se trata el algoritmo de QAOA en un ejemplo de aplicación a una combinatoria de espines de 3 partículas, aquí se explica como se lleva a cabo las dos secciones del algoritmo QAOA (la etapa de aplicación de los operadores cuánticos y la etapa de búsqueda de los parámetros "ideales" para los operadores). El primer enfoque que se realiza en el libro consiste en implementar búsqueda exhaustiva de las posibles combinaciones que pueden darse de los parámetros γ (para el operador de fase) y β (para el operador mezclador), este método es aplicable al problema que se plantea en el libro debido a la simplicidad del mismo, además, se toma a consideración que la combinación de parámetros solo se puede dar en ciertos "saltos" discretos de valores posibles (dado que en la realidad los parámetros son de tipo continuo y pueden tomar un rango de valores contenido entre 0 y 2π). El segundo enfoque que se implementa para este problema es la aplicación del algoritmo del gradiente descendiente (clásico) el cual es incorporado de manera tradicional, este enfoque permite realizar la obtención de los parámetros γ y β, dado que el gradiente se aplica como una función multivariante, sin embargo, como se puede observar (y como los autores señalan) este método carece del enfoque cuántico, es decir, que el algoritmo QAOA pierde su característica "cuántica" y se vuelve un algoritmo híbrido cuántico-clásico, además de mencionarse que este método solo puede ser aplicado para funciones de tipo convexo, ya que solo en esos casos el gradiente descendiente garantiza la obtención el mínimo global de la función (multivariante o no). [10]

Un método propuesto por *Ryan Sweke et al. (2020)* como alternativa al uso del algoritmo del descenso del gradiente clásico, es utilizar el modelo del algoritmo del descenso del gradiente estocástico (DGE), el cual hace uso del enfoque del gradiente estocástico el cual reemplaza las derivadas parciales exactas por un estimador de la derivada parcial, el algoritmo DGE es utilizado de manera convencional en problemas de gran escala en ML, donde se ha demostrado sus ventajas sobre los métodos de gradiente descendiente exacto, estas mejoras se dan en la evaluación más rápida del gradiente, mayor velocidad en la convergencia y la capacidad de evitar mínimos locales. Además, *Ryan Sweke et al. (2020)*, mencionan un trabajo de *Harrow and Napp (2019)*, el cual establece que existe

una interacción natural entre el método del DGE y los algoritmos de optimización cuántico-clásico, donde en particular existe una clase simple de problemas de optimización con los que se puede probar que DGE de "first-order" llegan a la convergencia de una forma más rápida que en cualquier método "zeroth-order". Finalmente, *Ryan Sweke et al. (2020)* llegan a la conclusión de que el descenso del gradiente exacto no es feasible para los casos de algoritmos híbridos clásico-cuántico (o cuántico-clásico), y que por ello, las perspectivas para tratar los problemas de descenso del gradiente deben hacerse desde el enfoque estocástico. [12] [13]

El uso de algoritmos del gradiente descendente (o métodos de Newton) clásicos, no son los únicos que se pueden aplicar para problemas de optimización presentes en algoritmos cuánticos, esto es importante dado que en este trabajo de investigación se busca establecer (sin importar el método o enfoque) que permita realizar con mayor eficiencia el proceso de búsqueda de los parámetros necesarios para QAOA. En el trabajo realizado por *Yao, J. et. al. (2020)* se presenta un algoritmo específico para realizar el cálculo de los parámetros de QAOA, este algoritmo hace uso del gradiente de política (policy-gradient) que es una técnica usada para aprendizaje por reforzamiento, esta técnica se puede entender como la aplicación de distribuciones de probabilidad que representan los parámetros que se buscan de QAOA, y lo que se busca es establecer la optimización en estas distribuciones de probabilidad, y a diferencia de los métodos del gradiente tradicionales (o no), esta técnica no necesita conocer las derivadas de la función, ya que estas son reemplazadas por un número finito de funciones de evaluación. Además, debido al funcionamiento del gradiente de política, que realiza la optimización en pequeños conjuntos, se puede evitar que el proceso de operación del algoritmo QAOA se vea afectado por distintas fuentes de ruido (principalmente decoherencia del hardware cuántico). Finalmente, *Yao, J., et al.* concluyen que el algoritmo del gradiente de política tiene muchas ventajas sobre los algoritmos del gradiente clásicos, esto debido a que permite realizar la optimización de los parámetros de QAOA sin la necesidad de saber explícitamente la información de las derivadas de la función, además, este método presenta una característica que no se había apreciado en otros algoritmos, y es que este método puede ser aplicado aun cuando la función objetivo no es suave con respecto al error, y finalmente se comenta que esta alternativa no solo es viable para el problema de QAOA, sino que también puede ser aplicable (en principio) a cualquier algoritmo tipo QVA (Quantum Variational Algorithm). [16]

Otro enfoque que trata el problema de optimización para parámetros en algoritmos cuánticos (en específico problemas de tipo QVA) se encuentra en el trabajo descrito por *Yamamoto, N. (2019)*, este trabajo realiza un análisis de la aplicación del algoritmo de gradiente natural (cuántico), la característica principal de este algoritmo es el uso de la geometría de la estructura del espacio de parámetros, esto es, la estructura que forman los parámetros a optimizar (la cual puede ser muy compleja, dependiendo de los parámetros que se busquen), ade-

más, en las aplicaciones de este algoritmo en problemas de inteligencia artificial se muestra que tiene una buena resistencia a los mínimos locales del espacio de parámetros. *Yamamoto, N.*, aplica el algoritmo de gradiente natural para el caso de un VQE (Variational Quantum Eigensolver), en este caso el gradiente natural actúa usando la forma paramétrica del estado cuántico, y con este, realiza un proceso de "elección" del mejor proceso de optimización de los parámetros del VQE, en el trabajo también se muestra la aplicación de esta propuesta para el caso de un qubit; donde se establece un desarrollo gráfico de los resultados los cuales demuestran la adaptabilidad del gradiente natural con respecto a otras técnicas, sin embargo, para el caso de 1 qubit no se muestra una gran mejoría, con esto *Yamamoto, N.* continúa con el caso de estudio de la obtención del estado base de una molécula de hidrógeno H^2, ahora para este caso cuando se observan los resultados de la energía contra el número de pasos (iteraciones) se muestra que el gradiente natural presenta una convergencia más rápida con respecto a los otros enfoques, y con esto se concluye que existen algunos problemas de optimización en los cuales el gradiente natural puede generar una mejor eficiencia del proceso de optimización, y esto se puede tomar como un buen resultado para poderse estudiar para el caso del algoritmo QAOA. [17]

Una metodología bastante interesante, la cual no se relaciona de manera directa con el uso del algoritmo del descenso del gradiente, es la realizada por *Shaydulin, R. et al. (2019)* la cual está basada en la optimización por multi inicio (multistart optimization). Como una premisa inicial, *Shaydulin, R.* explica las ventajas de los algoritmos híbridos clásico-cuántico, lo cuales presentan una posibilidad de ser aplicados en computadoras que se consideran computadoras NISQ, donde estas computadoras tienen la característica de ser de pocos qubits (alrededor de 100), y además en estos dispositivos existe un alto nivel de ruido y de perturbaciones externas por lo cual, las aplicaciones de alta demanda (como el algoritmo de Shor) no se pueden llevar a cabo actualmente, sin embargo, como lo explican en el trabajo, los algoritmos como QAOA y VQE son bastante feasibles para ser aplicados hoy en día. Algo que se explica también es que el método que proponen de optimización por multi inicios, busca también atacar el problema de la optimización de los parámetros para QAOA (en específico) aun cuando el espacio de parámetros sea de tipo no convexo, o contenga múltiples mínimos locales no degenerativos; esta consideración se rescata como importante debido a que muchos enfoques que utilizan el método del gradiente (sobre todo la metodología clásica) no contemplan este tipo de espacios paramétricos de los valores del algoritmo QAOA (o incluso del VQE). Ahora, retomando la metodología planteada por *Shaydulin, R.* consiste, como su nombre lo dice, en generar múltiples puntos de inicio para aplicar un método llamado APOSMM (Asynchronously Parallel Optimization Solver), este algoritmo consiste en establecer un punto de partida (aleatorio en teoría) donde a partir de este se realizan diferentes pruebas en paralelo, estas pruebas buscan encontrar los mínimos locales de la función y posteriormente obtener sus resultados para concluir cuál es el mínimo global. El

trabajo concluye que el método es más eficiente para administrar el número de evaluaciones prueba, ya que cada valor propuesto para los parámetros de QAOA se considera independiente en cierto momento, sin embargo, también se comenta que a medida que la profundidad (y con ello, el número de parámetros aumenta) la complejidad del problema aumenta demasiado, lo cual pudiera hacer que este método no sea tan conveniente de implementar en escalas más grandes que una computadora cuántica NISQ. [18]

1.5.2. Enfoques implementando machine learning para optimizar parámetros

Como se mencionó anteriormente, existen otros enfoques para tratar el problema de la optimización de los parámetros para el algoritmo QAOA (operadores de fase y mezcladores), este es el caso del artículo de *Alam, M. et al. (2020)*, este artículo presenta algo bastante interesante, y es que *Alam, M.* (y sus coautores) proponen un método para poder optimizar la obtención de los parámetros γ y β el cual incorpora Machine Learning (ML). De manera general, este enfoque es aplicado para analizar los patrones que se tienen respecto a la profundidad del algoritmo QAOA, esta profundidad se relaciona con la cantidad de operadores mezcladores (y de fase, en algunos casos) que son implementados para mejorar la eficiencia de la generación de los estados de superposición, esta profundidad permite que la etapa de evaluación combinatoria, es decir, la etapa en la que (de manera cuántica) se generan las combinaciones de estados (si los parámetros ingresados para γ y β son "buenos") que tienden a tener una mayor probabilidad de ocurrencia en las combinaciones que presentan la solución óptima de nuestro problema; sin embargo, el hecho de generar una mayor profundidad para QAOA genera la existencia de más parámetros γ y β, los cuales se presentan en pares (típicamente) por cada nivel de profundidad, y esto se traduce, de forma tradicional, a más parámetros a optimizar de forma clásica. En el trabajo de *Alam, M. et al. (2020)* se generó y entrenó un modelo usando ML, este modelo predice una relación que se presenta en modelos de QAOA de alta profundidad, usando el modelo entrenado se puede realizar una predicción sobre los parámetros iniciales para QAOA. Además de poder tener parámetros de salida proporcionados por el modelo, los datos indican que estos parámetros se acercan a los valores óptimos, lo cual de manera natural reduce el número de iteraciones para la búsqueda de las γ's y β's. Los resultados obtenidos por *Alam, M. et al. (2020)* indican que su método puede reducir desde un 44.9 % hasta un 65.7 % (dependiendo de los casos) el número de iteraciones necesarias para encontrar los parámetros óptimos para un QAOA de alta profundidad. [14]

Otros dos algoritmos que se presentan técnicas de ML para identificar los parámetros de QAOA son expuestos por *Khairy, S. et al. (2020)* quienes proponen dos modelos usando, para uno, Reinforcement Learning (RL), y para el otro, Kernel Density Estimation (KDE). *Khairy, S. et al.*, establecen algunos puntos

relevantes que se deben mencionar antes de abordar como se utilizaron estos métodos de ML para resolver el problema de optimización de parámetros de QAOA, el primer punto que rescatan es que la calidad de la solución o la aproximación que puede generar QAOA se relaciona de manera directa con los parámetros usados para sus operadores, en otras palabras, que visto desde el punto de vista estándar, el resultado de QAOA depende de la operación efectiva de los algoritmos de optimización clásicos, sin embargo, (como segundo punto) la optimización de los parámetros para QAOA (en la mayoría de los casos) se trata de un problema no-convexo con mínimos locales no degenerativos de "baja-calidad" (low-quality). Para el enfoque que consiste en utilizar RL, se genera una red neuronal "política" (policy), esta red permite analizar los datos geométricos (buscando patrones) de la función objetivo, con esta información se puede realizar una predicción "punto de partida" para modelos que tengan instancias nuevas. El enfoque que utiliza KDE representa la información del circuito cuántico como un modelo generativo (buscando su óptimo), este modelo ya entrenado puede ser aplicado de manera similar al de RL, inicializando nuevos parámetros y probando las instancias del problema en cuestión, para poder llegar a los parámetros óptimos. Finalmente, el artículo concluye que ambos métodos son aplicables al algoritmo de QAOA, dado que tanto el modelo de RL como de KDE (en el entendido de que tienen diferentes puntos aplicación en QAOA) mejoraron la eficiencia y la obtención de los parámetros para los operadores de fase y mezcladores, esto aunado a que el entrenamiento fue realizado con pequeños problemas y algunas instancias, permite pensar en gran potencial para tratar el problema de optimización de parámetros para algoritmos de QAOA de mediana profundidad, sin embargo, tiene que tomarse en cuenta que aun este método sea eficiente a pequeñas y medianas escalas, debido al tipo de problema de optimización que presenta la búsqueda de parámetros, se piensa como bastante improbable para poder ser usado en QAOA de alta profundidad. [20]

1.5.3. Enfoques cuánticos para optimizar parámetros

Actualmente, se siguen desarrollando distintos enfoques que utilizan algoritmos con métodos no tradicionales, que buscan resolver los problemas de optimización (presentes en muchos algoritmos cuánticos), sin embargo, para propósitos de este trabajo, ahora se comenzarán a analizar las propuestas que hacen uso específico de un algoritmo cuántico para resolver un problema de optimización que sea aplicable al caso del algoritmo QAOA.

En el artículo realizado por *Jordan, S. P. (2008)* se describe un algoritmo el cual busca obtener el gradiente, este gradiente tiene la característica de que es pensada como si fuese una "caja negra" (este término es ocupado en muchos algoritmos cuánticos para referirse al mecanismo de acción de una función f sobre una variable de entrada x que normalmente da un resultado $y \oplus f(x)$ donde y son qubits ancilla). Este método es conocido para algoritmos clásicos, sin embargo,

dado el funcionamiento del sistema clásico el método de utilizar funciones como cajas negras resulta bastante costoso al realizar las funciones de evaluación, pero para el caso cuántico *Jordan, S. P.* muestra que es posible realizar esta evaluación partiendo de una sola caja negra. El funcionamiento del algoritmo de *Jordan, S. P.* se asemeja al funcionamiento del algoritmo de Bernstein-Vazirani, otra diferencia es que el algoritmo busca establecer los valores de algunas variables enteras, esto es distinto a la aplicación de la caja negra convencional, donde se busca aproximar funciones. Finalmente, se concluye que el algoritmo se presenta como una opción viable para tratar cierto tipo de problemas de optimización o búsqueda de raíces, donde el aporte principal del algoritmo de Jordan consiste en transformar un problema de minimización de un funcional a un problema de minimización una función de muchas variables, además, se menciona que puede existir una mayor eficiencia en el proceso de obtención de las derivadas parciales haciendo uso de la ecuación de Euler-Lagrange. [11]

Continuando con el algoritmo propuesto por *Jordan, S. P. (2008)*, *Gilyén, A. et al. (2019)* continuaron con el trabajo de Jordan, realizando un algoritmo que mejora en eficiencia comparado con lo logrado anteriormente para el gradiente descendiente cuántico, este algoritmo también cuenta con la característica de ser multivariante, es decir, que puede utilizarse para optimizar múltiples parámetros, esto es algo importante porque como se mencionó anteriormente el algoritmo QAOA necesita de por lo menos dos parámetros γ y β, pero además, según la profundidad requerida por el algoritmo (principalmente se relaciona a la cantidad de operadores mezcladores) puede aumentar el número de parámetros necesarios para optimizar. Tomando esto en cuenta *Gilyén, A. et al.* prueba que se puede obtener una buena aproximación del gradiente con una mejor dependencia cuadrática de la precisión para la evaluación de la función. Finalmente, el trabajo concluye que si existe una mejora efectiva para una clase de funciones suaves, podría decirse que es un método más general y aplicable que el algoritmo inicial de Jordan, sin embargo, para casos de funciones de bajo grado multivariantes se puede generar una mejora en la velocidad de manera exponencial, esto resulta de mucha importancia e interés debido a como se describe el problema de optimización para el caso del QAOA donde posiblemente este enfoque puede resultar bastante eficiente para ser aplicado en sustitución del enfoque clásico visto de forma tradicional. [15]

Los métodos que realizan el uso de un enfoque cuántico para realizar las tareas de optimización (haciendo énfasis en su aplicación en el algoritmo QAOA) está el trabajo realizado por *Patrick Rebentrost, et al. (2019)* en el cual proponen un algoritmo de optimización iterativa que funciona específicamente para el gradiente descendiente y el método de Newton, el cual se postula como un método capaz de resolver problemas no cuadrático convexo e incluso problemas de optimización no convexos. La idea principal del algoritmo se ejecuta con las múltiples "copias" de un estado cuántico, en el entendido de que estas copias no son precisamente exactas, ya que de lo contrario violaría el teorema de no clona-

ción de la mecánica cuántica; pero pueden generarse estados muy similares donde se conoce que diferencias tienen y con esto poder llegar a reproducirlos de ser necesario (es decir, si ese estado resultara como el indicado en la optimización), estas "copias" se utilizan para producir múltiples "copias" de otro estado cuántico, esto se realiza considerando el vector gradiente y la matriz Hessiana de la función objetivo. En el caso en que se trata este método del gradiente cuántico, es especial para funciones de tipo polinomial con restricción de normalización. Sin embargo, *Patrick Rebentrost* comenta que debido al mecanismo de funcionamiento del algoritmo (copias del estado que convergen en uno solo), este proceso aumenta exponencialmente el número de operaciones necesarias, lo cual lo hace poco feasible para problemas en los cuales se necesite de un alto número de iteraciones para llegar a un resultado, pero debido a sus características este método puede ser incorporado como una solución en la búsqueda de mínimos locales, e inclusive siendo combinado con otro método como el APOSMM podría resultar en un algoritmo bastante robusto, considerando su aplicación para el algoritmo QAOA con alta profundidad. [19]

1.6. Computación cuántica

Una computadora cuántica (que es el objeto de estudio de la Computación Cuántica) se puede definir como: "A quantum computer is a device that leverages specific properties described by quantum mechanics to perform computations" [1], en pocas palabras, es un dispositivo que permite realizar procesos computables usando los principios de la mecánica cuántica para evolucionar el estado interno del sistema.

Todos los experimentos referentes a la mecánica cuántica (incluyendo al cómputo cuántico) busca respuestas (de forma numérica o analítica) usando probabilidad y estadística para describir los procesos que siguen las partículas (como fotones, protones, electrones, neutrones, etc.), estos procesos pueden o no tener presencia en fenómenos macroscópicos. Una de las principales restricciones que se presentan en la computación cuántica de manera particular, es el concepto de la medición que se explicará más a detalle en secciones posteriores, esta medición se repite de manera igualitaria en los experimentos preparados de manera repetitiva, esto genera que las mediciones tengan una "frecuencia relativa" (fr) y que la distribución de los valores cuánticos de las partículas se localice cerca del "valor promedio" (vp). [1]

El cómputo cuántico emplea los distintos principios presentes en la mecánica cuántica, siendo los más ocupados los de superposición, entrelazamiento, paralelismo cuántico, efecto de túnel cuántico, entre otros. [3]

El diseño algorítmico cuántico es bastante diferente al del diseño de algoritmos para una computadora clásica. Los algoritmos cuánticos tienen la característica de

ser mayormente basados en álgebra lineal y teoría de probabilidades, aún teniendo en cuenta que las ideas básicas; la mecánica cuántica pueden ser bastante confusa. La aplicación de una teoría basada en álgebra lineal y probabilidad hace que la computación cuántica sea más "amigable" para entender los fenómenos cuánticos.

Como se mencionó antes, el cómputo cuántico se enfrenta de manera directa con los elementos llamados **observables** que se relacionan con propiedades netamente cuánticas, ya sea el spin o polarización (para el caso del fotón), esto difiere de la mecánica cuántica como disciplina de la física que busca también agrega un peso de importancia a las variables de posición y momento de una partícula. [3]

- Una "medición" está asociada con una cantidad física llamada **observable**, estos observables pueden ser vistos (de manera matemática) como operadores auto-adjuntos en su forma matricial, y que están presentes en un espacio de Hilbert. [3]
- La **frecuencia relativa** se expresa como una probabilidad de obtener (medir) un estado durante un cierto número de mediciones.
- En mecánica cuántica se puede realizar una "predicción" que se obtiene del valor promedio de la medición de una cantidad física específica (observable) y se le conoce como **valor esperado**.
- Las mediciones y los valores promedio de los observables, tienen la tarea de describir el **estado** del sistema. Los estados están separados por dos clases, **estados puros** que pueden ser expresados como un vector en un espacio de Hilbert de una dimensión específica; y los **estados mixtos** que son operadores de tipo positivo, auto-adjunto y con valor $Tr(E_m) = 1$ (valor de la traza igual a 1).

El estado del sistema se entiende como un vector o un operador, estos elementos son en realidad una matriz (con características específicas) que opera su espacio de Hilbert, el espacio de Hilbert está definido como un espacio vectorial de números complejos, y la transformación o evolución del sistema (equivalentes a procesos de cómputo) son debido a matrices unitarias (para el caso de estados puros) aplicadas al mismo espacio de Hilbert. En conjunto, los estados y operadores, proveen un medio de cálculo de probabilidades y valores esperados para los distintos experimentos cuánticos realizados (circuitos computacionales cuánticos).

Definition 1.1. Un **espacio de Hilbert** expresado por $\mathbb{H}$ es un espacio vectorial complejo, con $\nu, \mu \in \mathbb{H}$ y $a, b \in \mathbb{C}$, donde a, b y $\nu, \mu \Rightarrow a\nu, b\mu \in \mathbb{H}$ también $a\nu + b\mu \in \mathbb{H}$. El **producto escalar** (positivo-definido), $\langle \cdot | \cdot \rangle : \mathbb{H} \times \mathbb{H} \to \mathbb{C}$. El producto escalar produce a la **norma** como $\|\cdot\| : \mathbb{H} \to \mathbb{R}$.

Una computadora clásica también puede describir procesos cuánticos, esto se logra cuando se aproximan datos mediante la simulación de la mecánica cuántica, pero las computadoras clásicas no utilizan las propiedades de la mecánica cuántica para desarrollar los cálculos de los procesos cuánticos simulados.

IBM tiene un servicio de cómputo a través de "cloud" que permite acceder a computadoras cuánticas reales con fines de académicos o de investigación, la experiencia IBM Q se basa en Qiskit que es un paquete de funciones que permite realizar, simular y medir circuitos cuánticos, el paquete Qiskit será usado en el presente trabajo para realizar la explicación y prueba de los algoritmos cuánticos. [5] [10]

En la computación cuántica, el principio de superposición es usado para representar a un "qubit" como una combinación lineal de los *estados base* (también conocidos como *estados fundamentales*) $|0\rangle$ y $|1\rangle$ (para la base computacional canónica). El qubit es la unidad computacional básica en cómputo cuántico, se puede entender de forma similar al bit clásico, sin embargo, un qubit está regido por los principios de la mecánica cuántica. [36]

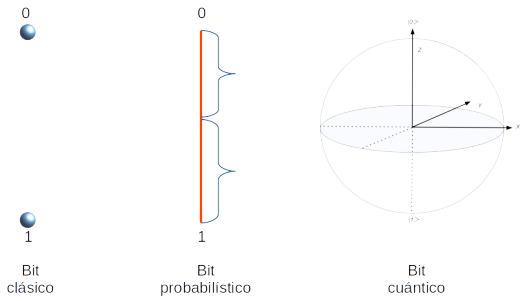


Figura 1.1: Representación de un bit clásico, probabilístico y cuántico.

Una de las formas más simples para entender el concepto de un qubit es interpretarlo como el estado de energía de un electrón en el átomo de hidrógeno.

Partiendo de la representación de la **Figura 1.2** donde el electrón es visto como un qubit, se tiene que:

Definition 1.2. Un qubit **simple** (visualizado en la **Figura 1.2**) está definido por solo dos estados (en superposición mayormente), posterior a una medición el estado toma el valor del estado base $|0\rangle$ o el valor del estado excitado $|1\rangle$ (los estados $|0\rangle$ y $|1\rangle$ se escriben usando la notación "bra-ket" de Dirac), estos estados son conocidos por el nombre de la **base canónica**, estos estados son vistos también como vectores que cumplen con las características de ser: unitarios $\|\psi\| = 1$ y ser ortogonales $\langle\psi|\phi\rangle = 0$, es decir, que los estados base son ortonormales.

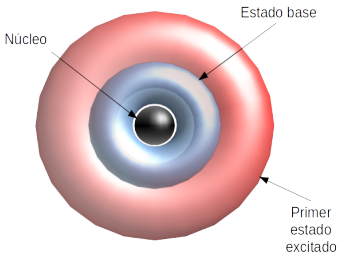


Figura 1.2: Representación cuántica del átomo de hidrógeno.

Con la **Definición 1.2** se puede establecer la representación de la base canónica en su forma vectorial "común". Para el caso del estado base (estado no excitado):

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{1.1}$$

Para el estado excitado:

$$|1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{1.2}$$

Como se mencionó anteriormente, el cómputo cuántico hace uso de los principios de la mecánica cuántica para llevar a cabo procesos computacionales, estos principios se resumen en:

- Superposición
- Evolución unitaria
- Principio de medición

Teniendo en cuentas estos principios podemos llegar a establecer otros fenómenos (no principios) como el **entrelazamiento**, **teletransportación cuántica**, **paralelismo cuántico**, entre otros.

Principios de superposición

El **principio de superposición** permite establecer una "combinación lineal de dos o más estados base que generan otro vector de estado dentro del mismo espacio de Hilbert, y este vector de estado describe el estado del sistema". [2]

En otras palabras, la superposición es un principio que permite describir un estado de sistema usando una combinación de los vectores base, esta representación está relacionada con uno de los aspectos más "contra intuitivos" de la mecánica cuántica en general, porque la superposición permite que el estado descrito se encuentre en más de un estado base al mismo tiempo, esto se conoce de manera popular como un estado que está en "dos posiciones al mismo tiempo".

Definition 1.3. Sea $|\psi\rangle$ la representación de un estado de sistema. Sea $\alpha \wedge \beta \in \mathbb{C}$, donde α y β son las **amplitudes** (dadas como números complejos) asociadas a los vectores base $|0\rangle$ y $|1\rangle$ respectivamente.

Usando la **Definición 1.3** sobre el principio de superposición, el estado $|\psi\rangle$ en la base canónica se puede representar como:

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \tag{1.3}$$

Y, agregando la notación vectorial estándar, el vector de estado está representado de la siguiente manera:

$$|\psi\rangle = \begin{bmatrix} \alpha \\ \beta \end{bmatrix} \tag{1.4}$$

Max Born (1926) demostró que la magnitud al cuadrado de los valores α y β pueden entenderse como probabilidades, y en cumplimiento con una regla importante en teoría de probabilidades los valores absolutos al cuadrado de α y β deben sumar 1. Algunas veces los vectores de estado del sistema deben volverse a normalizar para que cumplan con la condición mencionada.

$$|\alpha|^2 + |\beta|^2 = 1 \tag{1.5}$$

Este análisis hecho por Max Born tiene una relación directa con el segundo principio de computación cuántica que es la **evolución unitaria**, debido a que cualquier operador usado siempre será unitario y esto permite garantizar que el estado siempre sume a 1 independiente de la forma en cómo el estado evoluciona con el tiempo.

$$|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \tag{1.6}$$

$$|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \tag{1.7}$$

Cuando dos estados tienen la "misma" combinación lineal (como en las **Ecuaciones 1.6** y **1.7**) de vectores base y solo existe una diferencia en los signos

asociados (ya sea positivo o negativo), se dice que tienen una **fase relativa** diferente. En cómputo cuántico, la fase se puede interpretar como el ángulo entre el vector de estado hacia alguno de los ejes (y depende del inicio relativo del vector de estado). Estas fases relativas tienen un gran impacto en los fenómenos de interferencia constructiva y destructiva, este fenómeno es usado ampliamente en distintos algoritmos cuánticos.

Un ejemplo de este fenómeno de interferencia se puede ver como:

$$\frac{1}{\sqrt{2}}(|+\rangle + |-\rangle) = \frac{1}{2}\left[(|0\rangle + |1\rangle) + (|0\rangle - |1\rangle)\right] = \frac{1}{2}\left[2\,|0\rangle\right] = |0\rangle \tag{1.8}$$

En el ejemplo de arriba, el estado $|1\rangle$ genera una interferencia destructiva debido a que las fases tienen diferente signo, mientras que el estado $|0\rangle$ genera interferencia constructiva porque las fases poseen signo igual. Algunas veces el resultado de la suma de los estados de fase relativos puede cambiar la **fase global**, la fase global representa la fase completa del sistema y cuando es modificada usualmente existe un cambio de signo en el estado base.

Principio de Medición

El principio de medición establece que cualquier **medición** (o interacción) corrompe un sistema cuántico. Dado que antes de la medición del sistema, el estado se considera "desconocido", y este estado desconocido puede visualizarse como una superposición de los estados base del sistema.

De manera general, el operador de medición se relaciona con un operador Hermiciano que interactúa con un estado ψ para obtener una cantidad física medible del sistema. El operador Hermiciano tiene muchas características como que el inverso del operador es su transpuesta $O^{-1} = O^T$ y que sus eigenvalores (valores propios) son números reales.

El proceso de medición también puede interpretarse como una "proyección" hacia uno de los vectores base.

En la **Figura 1.3** explica de forma geometría el principio de medición (usando un sistema real de coordenadas), el estado del sistema denotado por ψ, donde ψ está representado como un estado en superposición.

$$|\psi\rangle = cos(\theta)\,|0\rangle + sin(\theta)\,|1\rangle \tag{1.9}$$

La idea de representar el principio de medición como una proyección es debido al hecho de que después de la medición la **función de onda**, que representa al estado del sistema, colapsa en alguno de los estados base. En el ejemplo de la **Figura 1.3** el estado colapsa en $|0\rangle$ o $|1\rangle$ con una cierta **probabilidad**, esta probabilidad se conecta con los valores de α y β (valores de amplitudes) en la

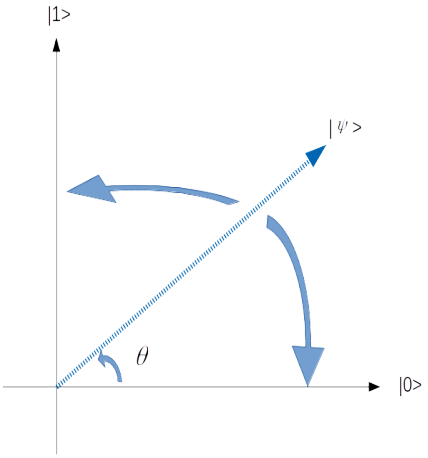


Figura 1.3: Representación geométrica del principio de medición en un sistema de coordenadas reales.

Definición 1.3. Usando el ejemplo geométrico presentado, el resultado después de la medición es:

- $|0\rangle$ con probabilidad $cos^2(\theta)$
- $|1\rangle$ con probabilidad $sin^2(\theta)$

Después de la medición del sistema, el nuevo estado del sistema se dice que se conoce de forma **determinista**, en caso de que el estado colapse a $|0\rangle$ el nuevo estado es $|\psi'\rangle = |0\rangle$ con probabilidad 1, por otro lado, si el estado colapsa a $|1\rangle$ el nuevo estado $|\psi'\rangle = |1\rangle$ con probabilidad de 1. Es importante mencionar que después del evento de medición (o cualquier otro tipo de perturbación externa al sistema) el estado cuántico pierde sus propiedades cuánticas. [4]

La medición del sistema es calculada u operada por el **producto interno** entre el vector de medición y el vector de estado.

Definition 1.4. En un sistema de k niveles (un espacio de Hilbert de k niveles) sea $|\psi\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle + ... + \alpha_{k-1} |k-1\rangle$ que sea la representación del estado del sistema y $|\varphi\rangle = \beta_0 |0\rangle + \beta_1 |1\rangle + ... + \beta_{k-1} |k-1\rangle$ la medición base, la proyección (probabilidad) de observar el vector $|\psi\rangle$ en $|\varphi\rangle$ es el **producto interno** visto como $(|\varphi\rangle, |\phi\rangle) = \langle\varphi|\phi\rangle$.

Con la **Definición 1.4** el producto interno tiene las propiedades (usando la **Interpretación de Copenhague**) de:

- Las amplitudes complejas $|\varphi\rangle$ son transformadas a sus correspondientes complejos conjugados y operadores como vector fila.
- Las amplitudes complejas $|\phi\rangle$ se mantienen iguales y se operan como vector columna.

Usando estas propiedades, desarrollando los elementos de los vectores de las siguientes ecuaciones, se tiene:

$$(|\varphi\rangle, |\phi\rangle) = \langle\varphi|\phi\rangle = \bar{\beta}_0\alpha_0 + \bar{\beta}_1\alpha_1 + ... + \bar{\beta}_{k-1}\alpha_{k-1} \tag{1.10}$$

En notación vectorial estándar:

$$\langle\varphi|\phi\rangle = \left[\bar{\beta}_0, \bar{\beta}_1, ..., \bar{\beta}_{k-1}\right] \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_{k-1} \end{bmatrix} \tag{1.11}$$

La probabilidad de obtener el resultado $|\varphi\rangle$ se obtiene con la siguiente ecuación.

$$Pr[\varphi] = |\langle\varphi|\phi\rangle|^2 \tag{1.12}$$

Como se mencionó anteriormente, después de la medición el nuevo estado del sistema es $|\psi'\rangle = |\varphi\rangle$.

El producto interno no es la única operación básica que se puede desarrollar dentro del espacio de Hilbert, el **producto externo** también tiene una mención importante, porque a diferencia del producto interno, el producto externo no genera mapas a los escalares, el producto externo permite crear las formas matriciales de los operadores.

Definition 1.5. Sea $|\psi\rangle$ el vector de estado en un espacio de Hilbert $\mathbb{H}$, y $|\phi\rangle$ un operador (posiblemente otro estado), el producto externo está definido como $\rightarrow |\phi\rangle \langle\psi| \Rightarrow operador$.

Usando la **Definición 1.5** se mostrará un ejemplo de aplicación del producto externo. En general, el producto externo puede ser visto como el "kernel" de las compuertas cuánticas, donde se trata de crear operadores unitarios a partir de operadores básicos aplicando producto externo, suma de vectores y resta de vectores.

$$|0\rangle \langle 0| = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \begin{pmatrix} 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \tag{1.13}$$

Principio de evolución unitaria

La evolución de un sistema cuántico se puede interpretar como la **rotación** del espacio de Hilbert en el que se encuentra (para el caso del cómputo cuántico) esto se realiza usando principalmente las "compuertas cuánticas". [4]

La rotación del espacio es realmente una **transformación lineal** y es similar (hasta cierto punto) a la rotación presente en el problema de cuerpo rígido, ya que todos los ángulos entre vectores de estado se preservan en la rotación. La transformación lineal se representa como un operador matricial en el espacio de Hilbert.

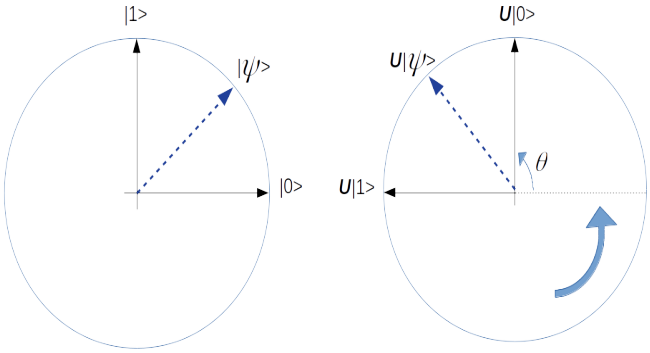


Figura 1.4: Evolución unitaria del estado $|\psi\rangle$ en $\mathbb{H}$

Como se muestra en la **Figura 1.4** la compuerta cuántica U (vista como una transformación lineal) aplica una rotación a todo el espacio de Hilbert, el operador cuántico $U = R_\theta$ puede describirse de la siguiente forma:

$$|0\rangle \overset{U=R_0}{=\!\!\rightarrow} cos(\theta)\,|0\rangle + sin(\theta)\,|1\rangle \tag{1.14}$$

$$|1\rangle \overset{U=R_0}{=\!\!\rightarrow} -sin(\theta)\,|0\rangle + cos(\theta)\,|1\rangle \tag{1.15}$$

Aplicando rotaciones individuales de R_θ a la base $|0\rangle$ y $|1\rangle$, el operador cuántico (rotación matricial) se describe como:

$$R_\theta = \begin{pmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{pmatrix} \tag{1.16}$$

Y para el caso de la transpuesta de la matriz de R_θ se denota:

$$R_\theta^T = \begin{pmatrix} cos(\theta) & sin(\theta) \\ -sin(\theta) & cos(\theta) \end{pmatrix} \tag{1.17}$$

Usando las **Ecuaciones 1.16** y **1.17** llegamos a la **Definición 1.6** que representa la transformación unitaria. Las transformaciones unitarias son el *kernel* de las compuertas cuánticas para la mayoría de las computadoras cuánticas disponibles hoy en día.

Definition 1.6. Sea I la representación de la **matriz identidad**, una transformación lineal u operador cuántico (compuerta cuántica) es $Unitaria \Leftrightarrow UU^T = U^TU = I$.

Un ejemplo de una transformación unitaria de 2×2 es $U = \begin{pmatrix} a & c \\ b & d \end{pmatrix}$ y el transpuesto conjugado de U es $U^T = \begin{pmatrix} \bar{a} & \bar{b} \\ \bar{c} & \bar{d} \end{pmatrix}$ generando $UU^T = U^TU = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$.

Paralelismo cuántico

Si bien el **paralelismo cuántico** no es un principio o postulado de la mecánica cuántica (y por ende, del cómputo cuántico) si es un fenómeno bastante interesante y complejo que ocurre durante la operación de un circuito cuántico. Este paralelismo se pretende explicar a continuación (se utilizará de forma arbitraria la notación de cómputo cuántico usando circuitos cuánticos, esta notación será explicada más a detalle en el transcurso del trabajo, por el momento solo se debe saber que el circuito debe ser leído de izquierda a derecha y que la compuerta U_f es una compuerta cuántica que cumple con todos los requisitos que se necesitan para su operación). [8]

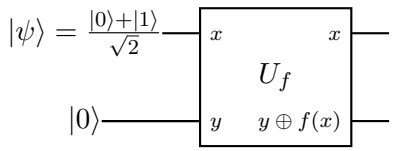


Figura 1.5: Paralelismo cuántico

En la **Figura 1.5** se tiene un esquema de un ejemplo del paralelismo cuántico, en el primer cable tenemos el estado $|\psi\rangle$ el cual entra como un estado en superposición y corresponde a la entrada x, abajo de este estado tenemos la entrada y a la cual ingresa el estado $|0\rangle$. Estos estados interactúan con la compuerta U_f, esta

compuerta mapea la salida (cable inferior derecho) al estado $f(x)$ si la entrada $y = |0\rangle$ y se tiene una operación módulo 2 en caso de que la entrada $y = |1\rangle$.

Lo interesante de este ejemplo de paralelismo se muestra cuando desarrollamos la aplicación de la entrada $|\psi\rangle$ y $|0\rangle$ con la compuerta U_f, esto nos da la siguiente ecuación:

$$\frac{|0, f(0)\rangle + |1, f(1)\rangle}{\sqrt{2}} \tag{1.18}$$

La **Ecuación 1.18** representa la superposición de la entrada $|\psi\rangle$ interactuando con y y U_f, esta superposición permite que en un mismo circuito (en un mismo tiempo) se tengan las evaluaciones para ambos valores $x \in (0, 1)$, y esto nos lleva a la posibilidad de evaluar dos o más condiciones (valores) de manera simultánea o en "paralelo" para una función $f(x)$ usando un solo circuito, esto es completamente distinto al "paralelismo clásico" ya que para evaluar una función (en una computadora clásica) se generan múltiples circuitos que se encargan de evaluar $f(x)$ para cada valor permitido.

El paralelismo cuántico está presente en muchos algoritmos cuánticos y resulta una de las principales ventajas sobre la computación clásica porque permite realizar evaluaciones de funciones con una mejora exponencial respecto a cualquier algoritmo clásico conocido. Algunos algoritmos que cuentan con este elemento de paralelización son: el algoritmo de Deutsch, el algoritmo Deutsch-Jozsa, el algoritmo de Shor, entre otros.

1.7. Entrelazamiento cuántico

Los principios de mecánica cuántica derivan en otros fenómenos contra intuitivos, el **entrelazamiento** es uno de los fenómenos más interesantes que se derivan de la teoría de la mecánica cuántica, el entrelazamiento puede verse como un caso especial del principio de superposición donde el estado general del sistema se describe por dos o más qubits, estos qubits muestran importantes características que no se pueden obtener de dos o más estados independientes.

"Quantum mechanical superposition called entanglement is when a measurement of one system (particle) is correlated with the state of the other system (other particle) in a way that is stronger than correlations in the classical world." [2]

La propuesta EPR busca demostrar que partículas pueden estar entrelazadas, y que la relación entre las partículas es instantánea, donde en principio la distancia entre las partículas no es importante, y cualquier tipo de alteración en alguna partícula (medición, interacción, etc.) afecta inmediatamente a la otra partícula. La paradoja EPR intenta explotar esta característica extraña de los

estados entrelazados para mostrar que la mecánica cuántica era "al menos una teoría incompleta".

1.8. Computación reversible

Casi todos los enfoques respecto a cómputo cuántico se basan en el cómputo reversible (circuitos reversibles), la idea detrás del uso de procesos de cómputo reversibles es evitar la pérdida de información que se da en cómputo irreversible. En computación clásica, algunas de las compuertas que se usan son compuertas irreversibles, como la compuerta NAND, OR inclusiva, OR exclusiva, etc. Estas compuertas pierden información porque diferentes entradas dan resultados idénticos, y al invertir el proceso no es posible recuperar la información completa de las entradas al inicio de la compuerta.

X	Y	X∨Y
0	0	0
0	1	1
1	0	1
1	1	1

Las operaciones reversibles permiten aplicar cualquier cantidad de operadores a los qubits mientras la coherencia del sistema se mantenga. En la presencia de ruido cuántico, la coherencia del sistema se puede perder. En computación cuántica reversible todos los operadores del circuito deben ser reversibles con la única excepción del operador **medición**.

1.9. Compuertas cuánticas unitarias

Las compuertas cuánticas que se aplican para realizar una transformación unitaria a un solo estado o qubit son unitarias, estos operadores son también Hermicianos, en otras palabras, los operadores son unitarios y reversibles (su operador inverso es su transpuesto).

Compuerta cuántica X

Una de las compuertas cuánticas básicas más importantes es la compuerta **cambio de bit** o compuerta **NOT** que se denota por el símbolo X y como su nombre lo indica, se encarga de transformar el estado de un qubit en su opuesto (ejemplo: $|0\rangle \xrightarrow{X} |1\rangle$).

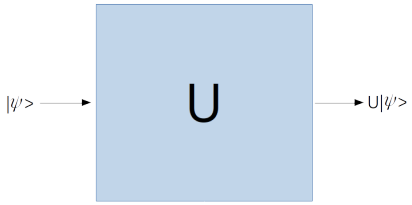
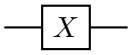


Figura 1.6: Unitary gate U applied to $|\psi\rangle \rightarrow U|\psi\rangle$

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \tag{1.19}$$

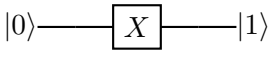


Como una mención interesante, cualquier compuerta u operador cuántico puede ser obtenido usando el producto exterior (véase en **Definición** 1.5), como un ejemplo de esto se obtiene la compuerta X usando producto exterior de la siguiente manera.

$$X = |0\rangle\langle 1| + |1\rangle\langle 0| = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \tag{1.20}$$

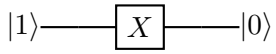
Ahora, para entender mejor los efectos de la compuerta X se aplicará en los estados base $|0\rangle$ y $|1\rangle$

$$X|0\rangle = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} = |1\rangle \tag{1.21}$$



Aplicando X al estado $|1\rangle$ tenemos que:

$$X|1\rangle = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} = |0\rangle \tag{1.22}$$



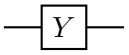
También la compuerta cuántica X puede ser aplicada a superposiciones de estados (no solo a estados base $|0\rangle$ o $|1\rangle$), para este caso las amplitudes asociadas a los estados base son intercambiadas después de la aplicación del operador.

$$\alpha |0\rangle + \beta |1\rangle \xrightarrow{X} \beta |0\rangle + \alpha |1\rangle \tag{1.23}$$

Compuerta cuántica Y

La compuerta cuántica Y también forma parte de las matrices de Pauli, este operador también es llamado σ_y, su función es generar rotaciones en el eje Y.

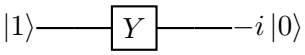
$$Y = \sigma_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \tag{1.24}$$



Como se puede observar en la **Ecuación 1.24**, el operador actúa en la parte imaginaria de las amplitudes de los estados, es decir, que el "eje Y" realmente es el eje imaginario.

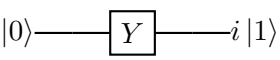
Algunos ejemplos de aplicación de esta matriz en la base de estados canónicos se muestra a continuación.

$$Y |1\rangle = \sigma_y |1\rangle = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} -i \\ 0 \end{pmatrix} = -i |0\rangle \tag{1.25}$$



Aplicando σ_y al estado $|0\rangle$.

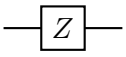
$$Y |0\rangle = \sigma_y |0\rangle = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} -i \\ 0 \end{pmatrix} = i |1\rangle \tag{1.26}$$



Compuerta cuántica Z

La compuerta cuántica de "cambio de fase" que se denota por Z o también por σ_z es otra compuerta básica unaria (es decir, que actúa sobre un qubit a la vez) y se encarga de realizar un cambio de fase en el qubit que es usada.

$$Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \tag{1.27}$$



De manera particular la compuerta Z solo afecta (en la base canónica) al estado $|1\rangle$ (así como el estado $|1\rangle$ en superposiciones), este fenómeno es diferente comparado con la compuerta X que permite modificar ambos estados base (y superposiciones) y esto es resultado de la forma en como la compuerta Z está definida.

$$Z|0\rangle = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} = |0\rangle \tag{1.28}$$

$$Z|1\rangle = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ -1 \end{pmatrix} = -\begin{pmatrix} 0 \\ 1 \end{pmatrix} = -|1\rangle \tag{1.29}$$

También, la compuerta Z puede actuar en superposiciones de estados, como el estado $|+\rangle$ o el estado $|-\rangle$, donde para esta base en particular la compuerta Z permite cambiar el estado $|+\rangle \to |-\rangle$ y del estado $|-\rangle \to |+\rangle$.

$$Z|+\rangle = Z\frac{1}{\sqrt{2}}|0\rangle + Z\frac{1}{\sqrt{2}}|1\rangle = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}\begin{pmatrix} 1 \\ 0 \end{pmatrix} + \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}\begin{pmatrix} 0 \\ 1 \end{pmatrix} = \cdots \tag{1.30}$$

$$\cdots = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ 0 \end{pmatrix} - \frac{1}{\sqrt{2}}\begin{pmatrix} 0 \\ 1 \end{pmatrix} = \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle = |-\rangle \tag{1.31}$$

Usando el operador X y el operador Z de manera conjunta podemos realizar operaciones de cambio de **fase global**, este operador se puede denotar como $Y_g = iXZ$.

Compuerta de rotación cuántica por ángulo θ

El principio usado para la compuerta cuántica de Pauli Z puede ser extendido a casos donde se busque realizar una rotación de fase con un ángulo en particular, este operador se escribe como R_θ.

$$R_\theta = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{pmatrix} \tag{1.32}$$

Se puede observar que Pauli Z es solo un caso especial del operador R_θ donde $\theta = \pi$, usando las identidades de Euler llegamos a que $e^{i\pi} = -1$ y con esto llegamos al caso del operador Z.

Existen otros dos casos particulares de R_θ que son importantes de mencionar. El primer caso es el operador S donde se asume que $\theta = \pi/2$, este operador rota el estado $90\measuredangle$ grados en el eje Z.

$$R_{\pi/2} = S = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix} \tag{1.33}$$

Otro operador llamado T realiza una rotación sobre el eje Z por un ángulo de $45\measuredangle$.

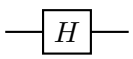
$$R_{\pi/2} = T = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix} \tag{1.34}$$

En adición, se puede observar que $S = T^2$, esto establece que realizar una operación T sobre el vector de estado representa la aplicación de $1/2$ de rotación de un operador S.

Compuerta cuántica Hadamard

La compuerta de **Hadamard** es una de las compuertas cuánticas más importantes porque **permite que un qubit que se encuentra en un estado definido $|0\rangle$ o $|1\rangle$ cambie a una superposición de los estados base**, la representación de la compuerta Hadamard es H.

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{1.35}$$



Esta compuerta fue desarrollada por el matemático John Sylvester que llamó a la compuerta en nombre de Jacques Hadamard. A continuación se puede observar la aplicación del operador H sobre los estados base canónicos.

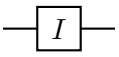
$$H|0\rangle = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}\begin{pmatrix}1\\ 0\end{pmatrix} = \frac{|0\rangle + |1\rangle}{\sqrt{2}} = |+\rangle \tag{1.36}$$

$$H|1\rangle = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}\begin{pmatrix}0\\ 1\end{pmatrix} = \frac{|0\rangle - |1\rangle}{\sqrt{2}} = |-\rangle \tag{1.37}$$

Matriz identidad

Otra compuerta cuántica que es bastante importante dado su significado en teoremas y resultados para garantizar las propiedades Hermeticidad de los operadores cuánticos, es el **operador identidad** que se representa por I, este no realiza ningún cambio en el qubit en el que es aplicado, en otras palabras, mantiene el estado actual del qubit operado.

$$I = \begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix} \tag{1.38}$$



Relaciones entre operadores para qubits simples

Una vez explicadas las compuertas básicas X, Y, Z, H y I, se pueden establecer relaciones bastante interesantes entre ellas. Estas relaciones permiten, para el caso de aplicaciones físicas de algoritmos cuánticos, tener una solución más cercana a las compuertas físicas disponibles en una computadora cuántica determinada, es decir, que permiten aproximar una compuerta cuántica que no se encuentra implementada físicamente en el dispositivo usando un conjunto de compuertas base que si existen internamente.

La primera relación que se puede mostrar, es la relación entre los operadores Hadamard y las compuertas Pauli X y Z.

$$HXH = Z \tag{1.39}$$

Donde también se tiene una relación inversa entre estas compuertas dada como:

$$HZH = X \tag{1.40}$$

Una relación interesante se da de la aplicación de las compuertas H y Pauli Y.

$$HYH = -Y \tag{1.41}$$

En este caso observamos que la fase global (signo del operador Y) se ve afectada, dando como resultado un operador Y con fase global rotada. Esta misma teoría se puede aplicar para compuertas cuánticas más específicas como las vistas para las de tipo R_θ.

1.10. Sistema de múltiples qubits

Los sistemas de múltiples qubits permiten representar sistemas cuánticos de mayor dimensionalidad que los qubits simples, en este caso cada estado del sistema es un vector normalizado (con amplitudes complejas), la normalización del vector de estado (en estados de múltiples qubits) garantiza que siempre se mantenga la evolución unitaria del sistema sin importar que compuerta sea aplicada, y esto a su vez garantiza que la probabilidad total sume a 1. [5]

Los sistemas de múltiples qubits se arman usando qubits individuales que son "juntados" usando **producto tensorial** denotado como $\otimes$, el producto tensorial es equivalente al producto de Kronecker de dos vectores.

$$|\psi\rangle = \begin{pmatrix}\alpha\\ \beta\end{pmatrix}; |\phi\rangle = \begin{pmatrix}\gamma\\ \delta\end{pmatrix} \tag{1.42}$$

$$|\psi\rangle \otimes |\phi\rangle = \begin{pmatrix}\alpha\\ \beta\end{pmatrix} \otimes \begin{pmatrix}\gamma\\ \delta\end{pmatrix} = \begin{pmatrix}\alpha\gamma\\ \alpha\delta\\ \beta\gamma\\ \beta\delta\end{pmatrix} \tag{1.43}$$

Cuando una medición es aplicada a los sistemas de múltiples qubits, cualquiera de los estados base (generados por el producto tensorial) son posibles como estados posteriores a la medición, donde cada estado tiene su probabilidad de aparición asociada a su amplitud compleja. Esta propiedad se relaciona con el crecimiento exponencial de los estados posibles en sistemas de múltiples qubits, este crecimiento está dado como 2^n donde n es el número de qubits del sistema.

$$|00\rangle = \begin{pmatrix}1\\0\\0\\0\end{pmatrix}; |01\rangle = \begin{pmatrix}0\\1\\0\\0\end{pmatrix}; |10\rangle = \begin{pmatrix}0\\0\\1\\0\end{pmatrix}; |11\rangle = \begin{pmatrix}0\\0\\0\\1\end{pmatrix} \tag{1.44}$$

En el ejemplo de la **Ecuación 1.44** se observa como el aumento de un qubit (dando como resultado un sistema de dos qubits) se tienen 4 posibles estados $|00\rangle$, $|01\rangle$, $|10\rangle$ y $|11\rangle$.

1.10.1. Operadores para múltiples qubits

De igual forma que los operadores para qubits simples, los sistemas de múltiples qubits tienen operadores característicos, en la mayoría de los casos este tipo de operadores se pueden ver como operadores condicionales, ya que usualmente relacionan la interacción de estados entre qubits para llevar a cabo una cierta operación.

Operador SWAP

El operador SWAP permite realizar "cambios" entre estados "intermedios", donde los estados "extremos" no se ven afectados.

$$SWAP = \begin{pmatrix}1&0&0&0\\0&0&1&0\\0&1&0&0\\0&0&0&1\end{pmatrix} \tag{1.45}$$

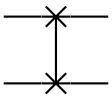
Usando los vectores base de la **Ecuación 1.44**, el operador SWAP permite que el estado $|01\rangle$ evolucione al estado $|10\rangle$.

$$SWAP\,|01\rangle = \begin{pmatrix}1&0&0&0\\0&0&1&0\\0&1&0&0\\0&0&0&1\end{pmatrix}\begin{pmatrix}0\\1\\0\\0\end{pmatrix} = \begin{pmatrix}0+0+0+0\\0+0+0+0\\0+1+0+0\\0+0+0+0\end{pmatrix} = \begin{pmatrix}0\\0\\1\\0\end{pmatrix} = |10\rangle \tag{1.46}$$

Ahora, si probamos aplicar este operador $SWAP$ al estado $|00\rangle$ veremos que el estado posterior al operador se mantiene como $|00\rangle$.

$$SWAP\,|01\rangle = \begin{pmatrix}1&0&0&0\\0&0&1&0\\0&1&0&0\\0&0&0&1\end{pmatrix}\begin{pmatrix}1\\0\\0\\0\end{pmatrix} = \begin{pmatrix}1+0+0+0\\0+0+0+0\\0+0+0+0\\0+0+0+0\end{pmatrix} = \begin{pmatrix}1\\0\\0\\0\end{pmatrix} = |00\rangle \tag{1.47}$$

La representación cuántica del operador $SWAP$ se da de la siguiente manera.



Operador CNOT

El operador $CNOT$ es uno de los operadores críticos en cuanto a computadoras cuánticas basadas en compuertas cuánticas se refiere, $CNOT$ permite realizar una operación NOT en algún qubit (llamado qubit "objetivo") usando otro qubit como activador (llamado qubit "control"). Usualmente, el primer qubit es el qubit de control y el segundo qubit es el qubit objetivo.

La compuerta $CNOT$ funciona usando el estado del qubit control, en caso de que el control se encuentre en $|0\rangle$ el qubit objetivo se queda inalterado por la compuerta, si control se encuentra en $|1\rangle$ se aplica un operador NOT de estado al qubit objetivo.

Uno de los principales usos de la compuerta $CNOT$ (en combinación con H) es generar estados entrelazados entre pares de qubits en un dispositivo cuántico.

$$CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{1.48}$$

Teniendo la representación de $CNOT$ en la **Ecuación 1.48** se puede aplicar esta compuerta a algunos estados para observar como se comporta.

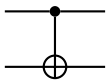
$$CNOT\,|00\rangle = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 1+0+0+0 \\ 0+0+0+0 \\ 0+0+0+0 \\ 0+0+0+0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} = |00\rangle \tag{1.49}$$

Como consenso general el qubit de la izquierda $|0_i\rangle$ en $|0_i 0_j\rangle$ es nuestro qubit de control y el de la derecha $|0_j\rangle$ es el objetivo, y se observa que dado el estado $|0\rangle$ de i el estado objetivo no se ve alterado y se mantiene.

$$CNOT\,|11\rangle = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0+0+0+0 \\ 0+0+0+0 \\ 0+0+0+1 \\ 0+0+0+0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} = |10\rangle \tag{1.50}$$

Ahora, observamos que el qubit de control i se encuentra en $|1\rangle$ y da paso a la aplicación de la compuerta NOT (interna de $CNOT$) al qubit objetivo j que se transforma como $CNOT\,|11\rangle \rightarrow |10\rangle$, es decir, que el estado del objetivo cambia de $|0\rangle \rightarrow |1\rangle$.

La representación del operador $CNOT$ en un diagrama de circuito cuántico es de la siguiente manera.



En diagrama anterior, el qubit de control está representado por el círculo en color negro (alambre cuántico superior) y el qubit objetivo esta representado por el símbolo $\bigoplus$ que es el qubit objetivo.

Algo interesante que se presenta entre las compuertas $SWAP$ y $CNOT$ (al igual que en las compuertas de Pauli y Hadamard), es que se puede generar una igualdad entre estos operadores de la siguiente manera (el qubit control es i y el qubit objetivo es j):

$$SWAP_{ij} = CNOT_{ij}CNOT_{ji}CNOT_{ij} \tag{1.51}$$

Operadores de control unitario

De manera general, cualquier operador unitario (X, Y, Z, etc.) puede ser combinado con un elemento de control, esto da lugar a una **compuerta de control unitario** que funciona de manera similar a la compuerta $CNOT$ con una diferencia en la evolución que se realiza al qubit objetivo, donde puede aplicarse un operador Z, Y, R_θ, etcétera.

$$CY = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -i \\ 0 & 0 & i & 0 \end{pmatrix} \tag{1.52}$$

En la **Ecuación 1.52** se observa un ejemplo de un operador de control unitario donde se usa una matriz Pauli Y y se agrega el "control" denotado como C.

$$CY\,|10\rangle = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -i \\ 0 & 0 & i & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0+0+0+0 \\ 0+0+0+0 \\ 0+0+0+0 \\ 0+0+i+0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ i \end{pmatrix} = i\,|11\rangle \tag{1.53}$$

Operador de Toffoli

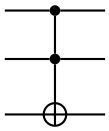
El operador de **Toffoli** o $CCNOT$ entra en la categoría de operadores ternarios, que realizan operaciones usando tres qubits, su operación es similar a la compuerta $CNOT$ con la diferencia de agregar un qubit de control extra para realizar la operación NOT en el qubit objetivo, pero en $CCNOT$ los dos qubits de control deben estar en el estado $|1\rangle$.

$$CCNOT(x, y, z) \mapsto (x, y, (z \oplus xy)) \tag{1.54}$$

El operador CCNOT expresado en su forma matricial está dado de la siguiente manera.

$$CCNOT = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \tag{1.55}$$

La representación del operador de Toffoli en circuitos cuánticos se da de la siguiente forma.



Operador de Fredkin

El operador o compuerta de **Fredkin**, también conocida como $CSWAP$ se puede interpretar como una compuerta $SWAP$ controlada, donde existe un qubit de control (que solo activa el operador $SWAP$ si está en $|1\rangle$) y dos qubits objetivo, estos qubit objetivo serán los que "intercambiarán" sus estados en lo que se refiere a la activación del control.

$$CSWAP = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{1.56}$$

1.10.2. Universalidad en computación cuántica

Como se sabe, existe un conjunto de operadores en computación clásica que en conjunto generan un grupo de operadores universales, es decir, que usando combinaciones de este grupo se pueden realizar cualquier tipo de operación clásica. El caso más simple de un conjunto de operadores universales es la compuerta clásica $NAND$ dado que está compuerta por sí sola es suficiente para reconstruir todos los operadores clásicos.

Para el caso de la computación cuántica también existen grupos de compuertas que generan universalidad.

- Compuerta cuántica Toffoli junto con un operador unitario de cambio de base con coeficientes reales (compuerta de Hadamard H).[6]
- La triada de compuertas $CNOT, T, H$. [7][8]
- Etc.

Otra forma de visualizar la universalidad de los operadores cuánticos es usando la **Esfera de Bloch** que se muestra en la **Figura 1.7**, esta esfera es usada principalmente para representar las operaciones realizadas sobre un qubit, ya que el estado del qubit se representa como un vector que parte del origen hasta un punto de la superficie de la esfera (unitaria). Hablando de la universalidad de las operaciones cuánticas, se pueden interpretar (para el caso de qubits simples o individuales) como operadores que permitan cubrir en su totalidad la esfera de Bloch.

1.11. Computadoras cuánticas

En lo que respecta a la implementación física de una computadora cuántica se debe mencionar que existen varios modelos y enfoques para realizarlas. Pero en este trabajo, se tomarán como referencia, en específico, el modelo de computadora

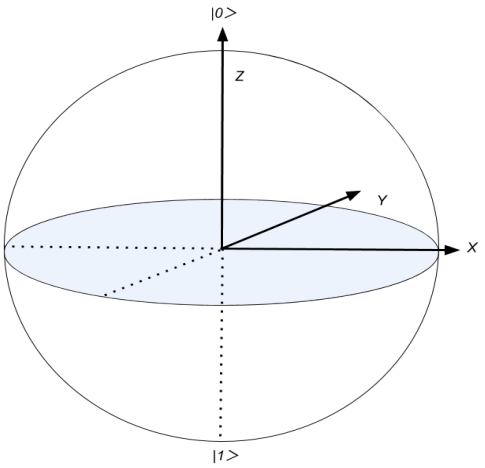


Figura 1.7: Esfera de Bloch

cuántica **basada en compuertas** (compuertas cuánticas) usando superconductores para la creación de los qubits, este tipo de tecnología es el que estudia IBM Quantum principalmente.

De manera general, las computadoras cuánticas del modelo basado en compuertas están conformadas por varios sistemas, el primer sistema es la etapa de control (que usa una computadora clásica) para controlar y monitorizar todos los procesos de preparación del circuito cuántico, temperatura, mediciones y presentación de resultados; después, se tiene la parte "importante" de la computadora, esta etapa se encuentran las herramientas para realizar la programación de circuitos cuánticos para correr algún algoritmo cuántico en específico, donde al ser un modelo basado en compuertas, cada qubit que se utiliza en el proceso se puede operar mediante compuertas cuánticas que son activadas (o desactivadas según sea el caso) en la preparación del circuito.

Ahora bien, dentro de este enfoque de compuertas cuánticas como base de una computadora cuántica existen varios aspectos que deben analizarse dado que existen muchas computadoras que utilizan este enfoque.

- **Universalidad**: Se refiere a que si la computadora cuántica en cuestión es universal o Turing-completa. Esta universalidad debe cumplir con varios factores, por ejemplo, si cuenta con las compuertas cuánticas básicas para considerarse universal, cumple con el criterio de DiVincenzo (relaciona la medición e interacción con qubits de forma individual), etc.
- **Fidelidad**: Esta característica se refiere a la capacidad que tiene la computadora de mantener la coherencia de los qubits (en operación), usualmente

se asocia con el resultado de $1 - \epsilon_r$ donde ϵ_r es la frecuencia de error. Es importante mencionar que la fidelidad puede variar dependiendo del tipo de compuerta que se use para observar la decoherencia del sistema, ya que una compuerta $CNOT$ que expande dos qubits presenta una mayor facilidad en perder su coherencia, a diferencia de los operadores de compuertas simples.

- **Escalabilidad**: Se refiere a que si la arquitectura permite un incremento de la capacidad de elementos que se pueden operar (qubits o compuertas cuánticas), sin perder su tolerancia de error.
- **Qubits**: En este rubro se prioriza el número de qubits que pueden ser usados por el sistema, además, también se analiza cómo están distribuidos esos qubits, ya sea si son adyacentes o no, y qué limitaciones presentan en distintas combinaciones u operaciones permitidas.
- **Profundidad de circuito**: Establece el número de operaciones posibles que se pueden realizar en la computadora cuántica antes de perder la coherencia del sistema.
- **Conectividad lógica**: La conectividad lógica se refiere a la capacidad de realizar operaciones entre qubits, dado que no todas las operaciones entre qubits pueden realizarse debido a su localización física en el sistema, esta conectividad se relaciona directamente con compuertas como $CNOT$, $SWAP$, $Toffoli$, etc.
- **Accesibilidad en CLOUD**: Esta característica es posiblemente la más importante de todas, ya que en su estado actual, tener acceso a computadoras cuánticas es bastante difícil, y por ello el acceso a dichos dispositivos (principalmente a través de servicios de cloud) resulta de vital importancia para considerar una implementación física de un circuito cuántico.

1.12. Programación cuántica

Las herramientas que permiten interactuar con hardware cuántico en la actualidad están basadas en programación usando lenguajes de alto nivel como Python, C/C++, Java, etc. Sin embargo, debido a su facilidad de aprendizaje y aplicación, en los últimos años la comunidad científica se ha enfocado más en el desarrollo de paquetes de cómputo cuántico usando Python.

Los lenguajes de programación cuánticos (QPL, Quantum Programming Languages) tienen como finalidad permitir la interacción del usuario en la parte superior de la **Figura 1.8**, esta interacción se da a través de instrucciones de bajo nivel que indican a la parte inferior (superconductores y compuertas) que compuertas cuánticas deben ser aplicadas a que qubits para implementar de forma correcta un circuito cuántico particular.

Figura 1.8: Idea general de una computadora cuántica

1.13. Modelo computacional cuántico

Existen varios modelos y vertientes de representación de procesos computaciones cuánticos. Actualmente, la representación más usada y útil, debido al nivel de desarrollo y estado tecnológico actual de las computadoras cuánticas, es el modelo de "circuito" computacional cuántico. Este modelo utiliza la noción de **compuertas cuánticas** (siendo estas reversibles y unitarias como se mencionó anteriormente) las cuales son aplicadas usando **cables**, estos cables son la representación abstracta del trayecto de los **qubits**, estos qubits son transformados con las compuertas cuánticas que pueden evolucionar el estado a uno o múltiples qubits, finalmente para poder obtener información comprensible y utilizable para el usuario, en la parte final de los cables cuánticos (normalmente) se coloca una compuerta especial que se encarga de realizar la **medición** del qubit en cuestión, es decir, que realiza mediciones uno a uno.

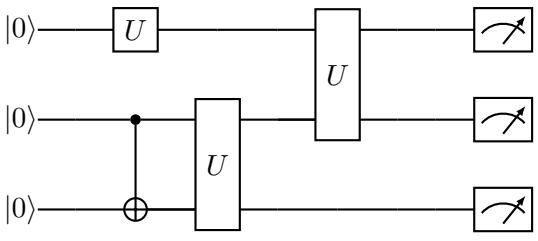


Figura 1.9: Modelo de circuito cuántico.

Observando la **Figura 1.9** se muestra como son las consideraciones generales para este modelo de computación cuántica. Inicialmente, se contemplan los qubits (lado izquierdo) inicializados (en la base computacional estándar) a $|0\rangle$, después cada qubit es transportado por su respectivo cable cuántico a lo largo de una serie de compuertas (leídas de izquierda a derecha) y finalmente, para extraer la información observable del qubit se usa la compuerta de medición (extrema derecha), que si no se especifica, siempre se considerará que la medición se realiza en la base computacional estándar.

Capítulo 2

Algoritmo de Aproximación de Optimización Cuántico

Los algoritmos cuánticos aplicados a problemas de "Inteligencia Artificial", "Optimización" o "Big Data" han tomado bastante interés en años recientes. Actualmente, el hardware cuántico disponible se considera de capacidad baja o intermedia, donde la mayoría de los dispositivos presentan fallas (debido al ruido, falta de control del sistema, etc.) y debido a estas fallas los algoritmos cuánticos se consideran solo en su parte teórica y no en su aplicación. [5] [2]

El caso del **Quantum Approximation Optimization Algorithm** (QAOA) presentado por *Farhi, E., et al.* (2014) [21], da una idea diferente, debido a que este algoritmo (en su forma convencional) es considerado como un híbrido "clásico-cuántico" que contiene etapas que se llevan a cabo desde el punto de vista de un algoritmo cuántico y otras etapas que son efectuadas desde el punto de vista clásico, pero esto no es lo único interesante de este algoritmo, y es que QAOA también es un algoritmo de "baja profundidad", es decir, que el algoritmo no necesita de una gran cantidad de qubits (solo los necesarios para representar el problema) y/o compuertas cuánticas para ser operado (esto conlleva que sea también de bajo tiempo de operación, lo que favorece su baja profundidad), estas características en conjunto permiten que QAOA sea un algoritmo con una buena resistencia a fallas y con un bajo nivel de decoherencia. [1] [9]

La **Figura 2.1** muestra la idea del funcionamiento del algoritmo QAOA a través de cuatro etapas, la primera etapa se refiere al *Estado inicial* en el cual se establece el número de qubits que representan al sistema donde todos están en el estado $|0\rangle$ (normalmente), después se tiene la etapa de *Aplicación de compuertas Hadamard* que modifica el estado original de los qubits dejándolos en un estado $|+\rangle$ que tiene una probabilidad igual de estar en el estado $|0\rangle$ o $|1\rangle$. Posteriormente, se pasan a las dos etapas que se asocian al funcionamiento de QAOA, la primera es la *Aplicación de Operador de Fase* que se encarga de generar la traducción del problema en una secuencia de compuertas cuánticas que formarán parte del circuito cuántico, estas compuertas generar una pequeña modificación a la probabilidad igualitaria de estados hecha por las compuertas Hadamard, pero

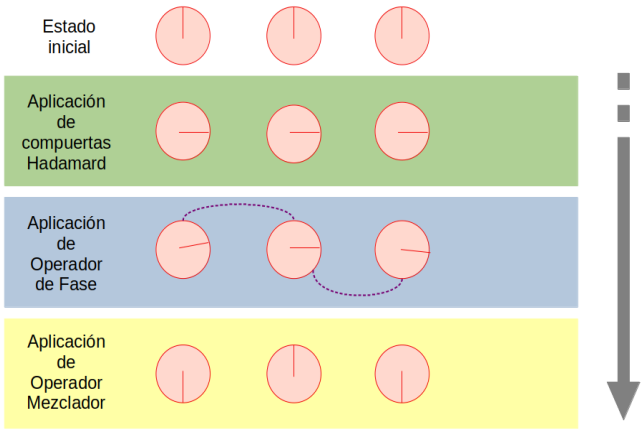


Figura 2.1: Representación ilustrativa del funcionamiento de QAOA.

más importante aún, el operador de fase genera las **conexiones** que existen entre las partículas (o nodos) del problema planteado, además, agregan las restricciones del problema (por ejemplo: para Modelo de Spin (Ising Spin Model) se tiene el campo magnético externo); la última etapa corresponde la *Aplicación del Operador Mezclador*, esta etapa genera en conjunto con el operador de fase los procesos de **interferencia** constructiva o destructiva, esto hace que exista (posterior al proceso de medición) una mayor probabilidad de medir ciertos estados sobre otros y se busca que al aplicar una combinación de hiperparámetros (parámetros que se usan para modificar a los operadores de fase y mezclador) correcta el estado (o los estados) óptimo (óptimos) tengan una mayor probabilidad de aparición.

QAOA se ha propuesto con mayor entusiasmo para problemas de optimización combinatoria, dichos problemas se presentan en distintas áreas que van desde la verificación de hardware hasta inteligencia artificial, algunos problemas comunes de optimización combinatoria son: el problema de la mochila (Knapsack problem), agente viajero, rutas de vehículos, etc. Estos problemas pueden ser clasificados en alguna de las siguientes categorías: Max-Sat, Max-Cut y Max-Clique. Estas categorías se generalizan con la siguiente representación: [1] [9]

Sea:

- Existen n variables de decisión binaria z_i.
- m funciones binarias (llamadas cláusulas) de las variables contenidas en $C_\alpha(z)$.
- Y $z_i \in \{0,1\}\, \forall i \in \{1,2,...,n\}$.

$$max : \sum_{\alpha=1}^{m} C_\alpha(z) \tag{2.1}$$

El reto de estos problemas es encontrar una asignación para z y que esta asignación maximice el número de cláusulas satisfechas. Estos problemas son de suma importancia para el cómputo científico, en específico, el problema Max-Sat se considera un problema NP-Duro (generalmente) que realmente es una variante (visto como problema de optimización) del problema de satisfacibilidad que es un problema NP-Completo.

Esta breve mención sobre los tipos de problemas de optimización combinatoria se da debido a que QAOA presenta su fuerte (hasta donde se conoce) en este tipo de problemas, y de hecho, cualquier algoritmo cuántico puede contar con bastantes ventajas en problemas combinatorios debido al paralelismo cuántico (algoritmo Deutsch-Jozsa, Transformada Cuántica de Fourier, etc.)

2.1. ¿Por qué el algoritmo QAOA?

QAOA se considera un algoritmo aplicable en un futuro cercano debido a sus muchas virtudes ya mencionadas, donde dichos factores generan que QAOA pueda ser usado en pequeñas computadoras cuánticas disponibles actualmente (como IBM Q, Google Quantum Computer, etc.), la aplicabilidad de QAOA permite que sea un algoritmo bastante popular y que se estudie de manera importante para resolver distintos problemas útiles, ya sea para observar la mejoría (o no) de este método sobre los métodos clásicos o para descubrir problemas en los que solo se puedan utilizar algoritmos cuánticos similares a QAOA para obtener soluciones óptimas. [35]

QAOA al ser un algoritmo de baja profundidad permite utilizar los recursos limitados que se tienen actualmente respecto a las computadoras cuánticas, y poder generar un resultado para un problema a escala (reducido) de un problema real, esto conlleva también que QAOA no necesite un tiempo prolongado de coherencia durante su aplicación, donde la coherencia se relaciona con la capacidad de mantener el estado cuántico del sistema sin que este se vea afectado por factores externos, y finalmente, existen resultados experimentales y teóricos que demuestran que QAOA (debido a su funcionamiento) es bastante resistente a errores por decoherencia y ruido dentro del sistema. [1] [9] [27]

Otro punto importante por el cual experimentar con QAOA, como una herramienta para resolver problemas de optimización, es su uso reducido de qubits, ya que en principio QAOA solo necesita un número de qubits equivalente al necesario para representar la información del problema.

2.2. Representación de una función de costo *C(z)* para ser usada en QAOA

Uno de los principales retos para la implementación de algoritmos cuánticos a problemas reales (cuando no se usan para problemas de mecánica cuántica o similares) es traducir la información del problema en algo que pueda ser operado por la computadora cuántica, usualmente esta traducción se da como una representación de la información del problema en un operador cuántico válido y posteriormente a un circuito cuántico, este operador cuántico debe contar con las características vistas en secciones anteriores. Una vez que la información del problema se convierte en un operador cuántico, este problema puede ser explorado usando distintos algoritmos cuánticos dependiendo de su naturaleza.

Usando el algoritmo QAOA se pueden resolver problemas que contengan un conjunto de variables que generen un gran espacio de combinaciones (problemas combinatorios), dentro del conjunto de combinaciones debe existir la combinación que de lugar al estado óptimo del sistema, y esta combinación en particular es la que optimiza algún criterio propuesto. Este criterio de optimización se relaciona con la **Función de Costo** denotada por $C(z)$, donde z contiene todo el conjunto de variables hasta n del problema de optimización combinatorio.

$$z \equiv \{z_1, z_2, ..., z_n\}\,; z_j\,\{-1, +1\} \tag{2.2}$$

Entonces teniendo la **Ecuación 2.2** el problema es como traducir $C(z)$ en un problema cuántico válido, esta "traducción" del problema consiste en crear un operador C que mapea la función de costo a un valor cuántico representativo. Este operador C se entiende como una matriz diagonal de $C(z)$, donde para el caso de QAOA la diagonal principal de la matriz contienen los valores de evaluación combinatoria para $C(z)$, es decir, los resultados de la aplicación de las combinaciones en la función del problema. [34] [35]

$$C(z) \rightarrow C\,|z\rangle = C\,|z_1 z_2 ... z_n\rangle = C(z)\,|z_1 z_2 ... z_n\rangle \tag{2.3}$$

La representación de la variable z_j puede ser expresada en cómputo cuántico usando el operador de Pauli Z, este operador cumple con la propiedad de que los valores de z deben ser binarios de la forma $\{+1, -1\}$, la forma del operador Z y el operador identidad I es de la siguiente manera:

$$Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}; I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \tag{2.4}$$

Con esta representación de Z en mente, ahora podemos incorporarla a la función de costo para obtener la nueva representación usando el operador Z, el operador identidad I y el producto tensorial de ellos.

$$C(Z) = Z_1 \otimes Z_2 \otimes ... \otimes Z_n \otimes I \tag{2.5}$$

Dependiendo del problema, la secuencia y como el producto tensorial de Z y I son aplicados será diferente.

$$C(Z) = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}_1 \otimes \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}_2 \otimes ... \tag{2.6}$$

$$... \otimes \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}_n \otimes \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{2.7}$$

Esta cadena de productos tensoriales entre los operadores Z e I resulta en la matriz diagonal de $C(z)$ de sus valores esperados.

$$\begin{pmatrix} C_{z_1} & 0 & 0 & 0 \\ 0 & C_{z_2} & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & C_{z_n} \end{pmatrix} \tag{2.8}$$

La **Ecuación 2.8** muestra la representación matricial que permite identificar donde aparecen los valores $C(z)$, donde cada valor i en $C(Z_i)$ corresponde a un valor de salida para la función de costo clásica. Una vez que la función de costo está representada como una matriz diagonal se procede a volverla una matriz unitaria (que se verá en la siguiente sección), ya que esto garantizará que esta sea un operador cuántico válido para poder ser optimizado mediante QAOA. [2] [3]

Teniendo lo anterior en cuenta, ahora se puede pasar a la aplicación de QAOA y explicar como se generan los operadores de fase y mezclador para cada tipo de problema, con esto se busca que se pueda tener una noción general de su aplicación y entender mejor estos operadores, su alcance y la teoría de aplicación detrás del funcionamiento de QAOA.

Capítulo 3

Desarrollo

En este capítulo se establecerá la metodología de implementación del algoritmo QAOA, en la primera parte se explicará como se traducen de los problemas de optimización combinatorios elegidos para este trabajo con una función de costo $C(z)$ a un operador cuántico, después se explicará de manera breve como los operadores de fase y mezclador operan (dando por sentada la revisión de la parte introductoria donde se explicó su funcionamiento más a detalle) y también se hablará de la profundidad planteada para estas implementaciones en específico. Posteriormente, se expresa como influye la variación de los parámetros γ y β (de los operadores de QAOA) y se hablará de los tres métodos que se implementarán, simularán y analizarán en el resto del trabajo.

3.1. Implementación de algoritmo QAOA

El primer paso a considerar para la implementación de QAOA es establecer el tipo de problema que se desea tratar, como se comentó anteriormente QAOA puede ser utilizado en problemas de optimización combinatoria, por ello para este trabajo se abordan dos problemas que se consideran dentro del área de los problemas combinatorios. El primer problema lleva por nombre **Ising Spin Model** o **Ising Energy Model**, que en su traducción al español se traduce como **Modelo de Ising** (para efectos prácticos se llamará a este problema como ISM por sus siglas en inglés), este problema consiste (de manera general) en establecer la combinación correcta de los spins o espines de las partículas del sistema de tal forma que esta combinación de lugar al estado (o estados según sea el caso) de mínima energía del sistema. El segundo problema lleva por nombre **Max-Cut**, que traducido al español se tiene el nombre de **Corte-Máximo** (de igual forma que antes, se llamará Max-Cut a los problemas de este estilo), este problema busca crear dos subconjuntos (a partir de un grafo que es el conjunto origen) donde estos subconjuntos tengan el mayor número de conexiones entre ellos, es decir, encontrar la combinación correcta para separar los nodos de un grafo en dos conjuntos cuyas conexiones entre conjuntos sean máximas. [27] - [32]

3.1.1. Descripción para problema ISM

Empezando con la descripción del primer problema, se puede observar el siguiente diagrama.

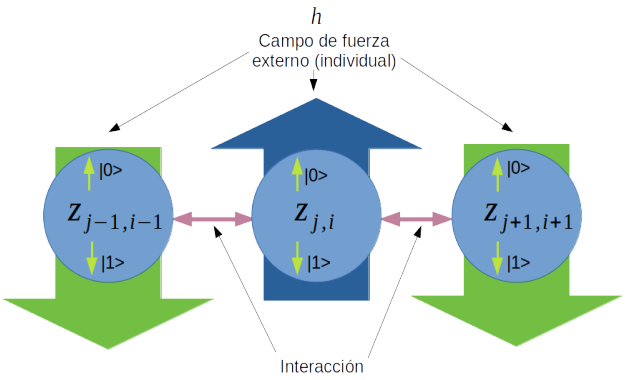


Figura 3.1: Representación del problema **ISM**.

Desglosando lo que se muestra en la **Figura 3.1** tenemos tres partículas representadas por los círculos color azul claro, estas partículas internamente se les asocia con la letra Z, esta letra tiene índices asociados j, i que sirven para identificar el número de partícula que se representa en el arreglo del sistema, además, estas partículas internamente pueden estar en un estado $|0\rangle$ (flecha interna hacia arriba) o $|1\rangle$ (flecha interna hacia abajo) este estado interno nos permite representar los "valores" válidos descritos en la **Ecuación 2.2** que se describen como $\{+1, -1\}$. Otro elemento que se observa de esta imagen son las flechas color verde o azul que se encuentran por detrás de las partículas, estas flechas se entienden como el campo de magnético (o fuerza) externo (individual) que tienen el símbolo h, estos campos de fuerza externos pueden estar orientados en dos sentidos (al igual que los spines de las partículas) por ello pueden ser de color verde o azul para diferenciar el sentido que estos tienen, usualmente este "sentido" se asocia al signo del campo externo. El último elemento de la figura **Figura 3.1** es la "interacción", esta interacción se presenta como la comunicación o dependencia de la conexión entre partículas vecinas, en otras palabras, que el Hamiltoniano que representa este problema también depende de la interacción entre las partículas vecinas.

La representación de la **Función de Costo** (como función de costo magnética de interacción y campo magnético externo) para el problema de ISM se muestra en la siguiente ecuación.

$$C(Z) = -\sum_{<i,j>} Z_i Z_j - \sum_i h_i Z_i \tag{3.1}$$

El tipo de problemas que describe la **Figura 3.1** y la **Ecuación 3.1** se presentan con bastante frecuencia en la naturaleza (principalmente en el estudio de configuraciones de partículas), estos sistemas buscan encontrar una configuración de partículas que permita alcanzar el estado de mínima energía del sistema, este estado se puede alcanzar cuando las partículas se encuentran en la combinación adecuada de spines $\{+1, -1\}$ (que corresponden a tener los estados $|0\rangle$ o $|1\rangle$). Esta descripción se puede entender como un híbrido entre la función de costo tradicional y una referencia (usando los operadores Z) a las matrices de Pauli tipo σ_z (pero estas aún no son implementadas como tal), esta descripción considera la interacción entre partículas (primer termino de la ecuación $-\sum_{<i,j>} Z_i Z_j$) y el aporte del campo magnético externo por partícula (segundo término de la ecuación $-\sum_i h_i Z_i$). Además, esta ecuación es el primer paso para traducir esta descripción al operador de fase de QAOA. El trabajo de QAOA es predecir esta configuración de mínima energía del sistema. [30]

Seguido de la **Ecuación 3.1** se deriva la siguiente representación, esta forma de la función $C(Z)$ se relaciona con la etapa del operador de fase dentro de QAOA.

$$C(U, \gamma) = e^{i\gamma \sum_{<i,j>} Z_i Z_j} e^{i\gamma \sum_i h_i Z_i} \tag{3.2}$$

$$C(U, \gamma) = \prod_{<i,j>} e^{i\gamma Z_i Z_j} \prod_i e^{i\gamma h_i Z_i} \tag{3.3}$$

Ahora con la **Ecuación 3.3** vemos como se transforma la función de costo $C(Z)$ (de la **Ecuación 3.1**) en un operador de fase válido para ser utilizado dentro de QAOA.

$$C(U, \beta) = \prod_i e^{i\beta X_i} \tag{3.4}$$

En la **Ecuación 3.4** tenemos establecida la segunda etapa del QAOA que corresponde al operador mezclador. Este operador, a diferencia del operador de fase, busca crear la interferencia constructiva y destructiva, ya que el problema inicialmente cuando pasa por el operador fase se puede considerar solo como la traducción de la función de costo al operador fase, mientras que el operador mezclador permite que ciertos estados prevalezcan sobre otros, es decir, aumentar o disminuir las probabilidades de ocurrencia de algún estado (u estados). [28]

3.1.2. Descripción del problema Max-Cut

El problema de Max-Cut se puede definir como un problema de partición combinatorio, la idea del problema es generar dos conjuntos (o subconjuntos dependiendo de como se aborde el problema) de nodos que contengan a los elementos de un grafo, estos conjuntos deben buscan cumplir la condición de que las conexiones entre nodos de diferentes conjuntos sean máximas. [27] [29] [31] [32]

Tomando como ejemplo un grafo de cuatro elementos, donde sus elementos están conectados de la siguiente forma (ninguna conexión tiene un peso asociado).

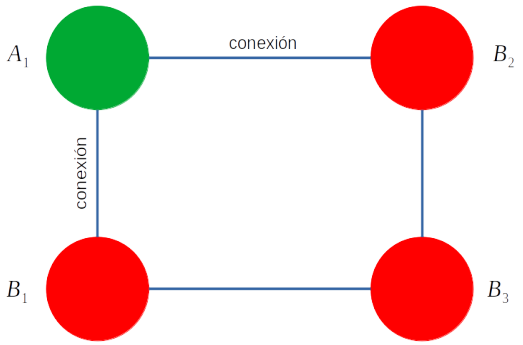


Figura 3.2: Problema **Max-Cut** para un grafo de 4 nodos con dos conexiones entre conjuntos.

En la **Figura 3.2** se tiene la representación del grafo ejemplo para entender Max-Cut, en la figura se aprecian dos grupos de nodos dados por el color verde para el grupo A y el color rojo para el grupo B, en este ejemplo se contempla que solo el nodo superior izquierdo $\{A_1\}$ pertenece al grupo A y los demás nodos $\{B_1, B_2, B_3\}$ pertenecen a B, esto genera que solo existan dos conexiones entre los elementos de A a los elementos de B (estas conexiones están escritas en la figura).

Continuando con el ejemplo, ahora tenemos la **Figura 3.3** que muestra otra configuración posible para separar el grafo en los conjuntos A y B, el conjunto A ahora contiene los elementos $\{A_1, A_2\}$ y B contiene $\{B_1, B_2\}$, sin embargo vemos que al igual que en caso anterior las conexiones que existen entre el conjunto A y B siguen siendo solo dos.

Finalmente en la **Figura 3.4** se aprecia una de las combinaciones correctas que nos garantizan el máximo de conexiones posibles entre los conjuntos A y B, igual que en la figura anterior cada conjunto tiene dos elementos, sin embargo, estos elementos se encuentran "intercalados" dentro de la distribución del gra-

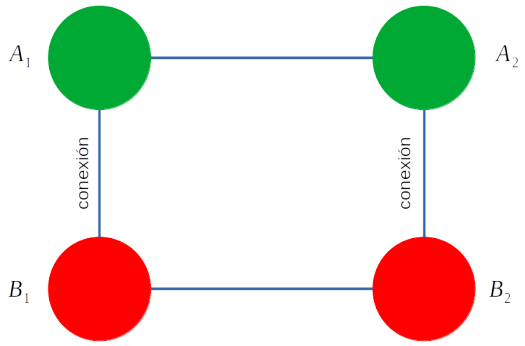


Figura 3.3: Problema **Max-Cut** para un grafo de 4 nodos con dos conexiones.

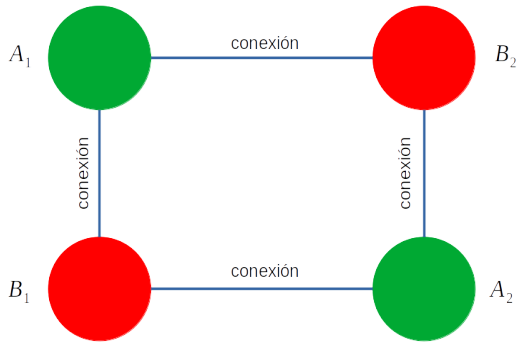


Figura 3.4: Problema **Max-Cut** para un grafo de 4 nodos con cuatro conexiones.

fo, en otras palabras, no se tienen nodos vecinos en un mismo conjunto, esto permite que en el ejemplo presentado se tenga el mayor número de conexiones posibles. La representación de como se observan estas conexiones dentro de un algoritmo pueden variar, pero debido a que se usará QAOA para poder realizar la búsqueda del estado óptimo de un problema similar, el estado que se asocia a esta configuración puede ser descrito como $|\psi\rangle = |0101\rangle$ donde los elementos en 1 corresponden a un conjunto y los elementos en 0 corresponden a otro, y como se mencionó anteriormente, existe otra configuración posible que garantiza el máximo de conexiones y es el inverso dado por $\left|\bar{\psi}\right\rangle = NOT\,|0101\rangle = |1010\rangle$.

De igual forma que el problema de ISM, el problema de Max-Cut utiliza QAOA y su evolución dada por el operador $U(\gamma, \beta)$, y de igual forma QAOA busca los parámetros óptimos para este operador denotados como $(\gamma_{opt}, \beta_{opt})$.

$$|\psi(\gamma,\beta)\rangle = \left(\sum_{i=0}^{n} U(\beta_i)U(\gamma_i)\right)|\psi_0\rangle \tag{3.5}$$

La **Ecuación 3.5** nos da otra forma de representar la aplicación de los operadores de fase $U(\gamma_i)$ y mezcladores $U(\beta_i)$ a nuestro estado inicial $|\psi_0\rangle$. El operador de fase o también conocido como **Hamiltoniano Problema** H_P se utiliza para poder realizar la traducción del problema Max-Cut (igual que para ISM).

$$U(H_P) = e^{-i\gamma H_P} = \prod_{<i,j>} e^{-i\gamma Z_i Z_j} \tag{3.6}$$

En la **Ecuación 3.6** se muestra la descripción del Hamiltoniano como el operador de fase (o Hamiltoniano Problema), esta descripción es bastante similar a la vista en la **Ecuación 3.1**, pero con la diferencia de que no existe un elemento que aporte un campo externo al sistema, para este tipo de problema Max-Cut donde no se consideran los pesos de las conexiones entre los nodos, el Hamiltoniano solo describe la interacción (conexión) entre los nodos vecinos.

$$U(H_B) = e^{-i\beta H_B} = \prod_{i} e^{-i\beta X_i} \tag{3.7}$$

Y al igual que en el caso del problema anterior, Max-Cut también necesita de un operador mezclador H_B dado por la **Ecuación 3.7** el cual funciona de manera idéntica al aplicado para el problema ISM, dado que este operador aplica una matriz X (operador **NOT** cuántico) para generar los procesos de interferencia constructiva o destructiva de los estados generados por el operador de fase. [28]

3.2. Instancias de los problemas para usar QAOA

Como una etapa previa a la descripción de los métodos de optimización que se probarán junto con QAOA es necesario especificar las características de los problemas a tratar.

Los problemas que se abordarán se mencionaron en la sección anterior, estos problemas son llamados **ISM** (o Modelo de Ising) y **Max-Cut**. Iniciando con el problema de ISM, se abordarán los problemas con 3, 4 y 5 partículas con las configuraciones (en términos de grafos) lineal, cíclico y completo. Estos problemas tendrán un campo magnético externo asociado a cada partícula con un valor elegido (incluido su signo, que representa la orientación del campo) de forma arbitraria.

3.2.1. Configuración lineal para ISM

Los problemas de tipo **lineal** se definen como una configuración de partículas en la que existen conexiones de partículas uno a uno, donde la primera y última partícula no tienen conexión más que con su vecino interno (para el caso de la primera partícula) y su vecino previo (para el caso de la última partícula).

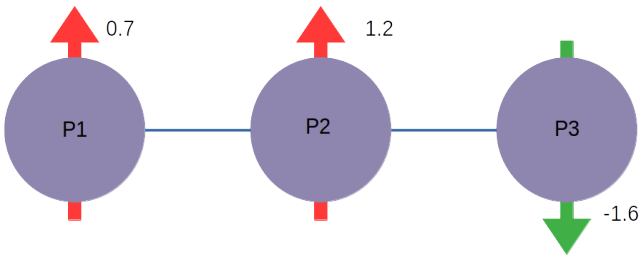


Figura 3.5: Problema **ISM** para un arreglo lineal de 3 partículas.

Observando la **Figura 3.5** se muestra la representación de la primera configuración lineal para el caso de 3 partículas, en esta figura se muestran las conexiones (interacciones) que existen entre las partículas $P1$ a $P2$ y $P2$ a $P3$, los campos magnéticos externos por partícula h se muestran con las flechas en color rojo o color verde, el color depende de la orientación que siente cada partícula, para el caso de $P1$ su campo es de $h_1 = 0.7$, para $P2$ su campo es de $h_2 = 1.2$ y para $P3$ su campo es $h_3 = -1.6$.

Dada la simplicidad del problema se puede establecer una tabla de estados en donde se colocan todos los posibles resultados según las combinaciones presentes en el spin de las partículas (recordando que solo pueden orientarse hacia $|0\rangle$ o $|1\rangle$), la energía del estado se puede calcular usando la **Ecuación 3.1** que básicamente establece la suma de las interacciones entre vecinos que será $+1$ si partículas vecinas tienen diferente spin, por ejemplo: si el estado de $P1$ es $|0\rangle$, el estado de $P2$ es $|0\rangle$ y el estado de $P3$ tiene estado $|1\rangle$ que se resume como $|0\rangle_1 \otimes |0\rangle_2 = |00\rangle_{12}$; y será -1 si las partículas vecinas tienen mismo spin, que se resume como $|0\rangle_2 \otimes |1\rangle_3 = |01\rangle_{13}$. Esto solo afecta a partículas que tengan una conexión directa y sean vecinas (existe interacción), y para el caso del campo magnético externo este se divide en dos casos, para el caso cuando la partícula está en $|0\rangle$ sí el campo es *positivo* (flecha color rojo) la partícula aporta $(-1) * (h_i) * (Z_i)$ y si el campo es negativo (flecha color verde) la partícula aporta $(-1) * (-h_i) * (Z_i)$, para el caso de la partícula en estado $|1\rangle$ si el campo es positivo (flecha roja) la partícula

P1	P2	P3	Costo
0	0	0	-2.3
0	**0**	**1**	**-3.5**
0	1	0	4.1
0	1	1	-1.1
1	0	0	1.1
1	0	1	-0.1
1	1	0	3.5
1	1	1	-1.7

Tabla 3.1: Combinaciones para problema ISM **lineal** para 3 partículas.

aporta $(-1) * (-h_i) * (Z_i)$ y si el campo es negativo aporta $(-1) * (h_i) * (Z_i)$.

Como se muestra en la **Tabla 3.1** el estado que tiene la mínima energía es $|001\rangle$ con un costo lineal de -3.5 (en negrillas), este estado también es el considerado óptimo para minimizar la función de costo original del sistema.

El siguiente paso es establecer cuál sería la ecuación que describe el problema mostrado (operador de fase), esta ecuación se deriva del uso de la **Ecuación 3.3** tomando en cuenta de los valores asignados para h_i así como las interacciones entre partículas.

$$U(C,\gamma) = \left(e^{i\gamma Z_1 Z_2} e^{i\gamma Z_2 Z_3}\right)\left(e^{i\gamma h_1 Z_1} e^{i\gamma h_2 Z_2} e^{i\gamma h_3 Z_3}\right) \tag{3.8}$$

De igual forma usando la **Ecuación 3.4** podemos obtener la ecuación que representará al operador mezclador para el problema de 3 partículas.

$$U(C,\beta) = e^{i\beta X_1} e^{i\beta X_2} e^{i\beta X_3} \tag{3.9}$$

Y con el operador mezclador desarrollado en la ecuación anterior, se puede continuar con los pasos de diseño del circuito cuántico y posteriormente las simulaciones locales y cuánticas para extraer y analizar los resultados obtenidos.

3.2.2. Configuración cíclica para ISM

Los problemas de tipo **cíclico** son bastante similares a los lineales con la diferencia que la última conexión se da entre la última partícula y la primera, y con esto se logra "cerrar" el ciclo. De igual forma que en el caso lineal, se comienza con la descripción del problema con 4 partículas.

El problema se muestra en la **Figura 3.6**, en la imagen se puede apreciar la configuración con 4 partículas llamadas $P1$, $P2$, $P3$ y $P4$, las cuales se encuentran conectadas a través de un ciclo donde $P1$ se conecta con $P2$, después $P2$ se

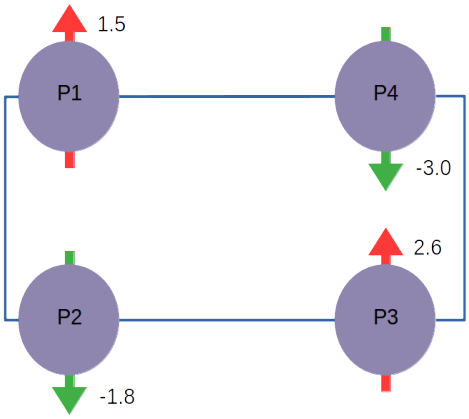


Figura 3.6: Problema **ISM** para un arreglo cíclico de 4 partículas.

conecta con $P3$, $P3$ se une a $P4$ y finalmente $P4$ se conecta con $P1$ para cerrar el ciclo. Los campos magnéticos externos asociados a cada partícula tienen valores asignados que van para $P1$ un campo es de 1.5, para $P2$ su campo es de -1.8, para $P3$ su campo es de 2.6 y para $P4$ su campo es de -3.0.

Usando la **Tabla 3.2** se observan las combinaciones y costos (de energía) de cada configuración posible, donde el estado de mínima energía está dado por $|1101\rangle$ con valor de -5.9.

También usando la **Ecuación 3.3** podemos describir la sección que corresponde al operador de fase del circuito cuántico del problema.

$$U(C,\gamma) = \left(e^{i\gamma Z_1 Z_2} e^{i\gamma Z_2 Z_3} e^{i\gamma Z_3 Z_4} e^{i\gamma Z_1 Z_4}\right) \left(e^{i\gamma h_1 Z_1} e^{i\gamma h_2 Z_2} e^{i\gamma h_3 Z_3} e^{i\gamma h_4 Z_4}\right) \quad (3.10)$$

La **Ecuación 3.10** es muy similar a la obtenida en la **Ecuación 3.8**, donde ambas ecuaciones solo difieren por la interacción de la partícula extra $P4$ y la conexión entre las partículas $P1$ a $P4$ que está presente solo para el problema cíclico. Para la sección que describe el operador mezclador, tenemos la siguiente forma:

$$U(C,\beta) = e^{i\beta X_1} e^{i\beta X_2} e^{i\beta X_3} e^{i\beta X_4} \quad (3.11)$$

La **Ecuación 3.9** y la **Ecuación 3.11** son bastante similares con la excepción del ingreso de la partícula $P4$, esto debido a que los procesos de interferencia se dan de forma similar entre el problema lineal y el cíclico.

P1	P2	P3	P4	Costo
0	0	0	0	-3.3
0	0	0	1	-5.3
0	0	1	0	5.9
0	0	1	1	-0.1
0	1	0	0	-2.9
0	1	0	1	-4.9
0	1	1	0	2.3
0	1	1	1	-3.7
1	0	0	0	3.7
1	0	0	1	-2.3
1	0	1	0	12.9
1	0	1	1	2.9
1	1	0	0	0.1
1	**1**	**0**	**1**	**-5.9**
1	1	1	0	5.3
1	1	1	1	-4.7

Tabla 3.2: Combinaciones para problema ISM **cíclico** para 4 partículas.

3.2.3. Configuración completa para ISM

La instancia del problema de configuración completa se refiere al término dentro de teoría de grafos donde todos los nodos (en este caso partículas) se encuentran conectadas unas con otras, es decir, que en el problema de ISM todas las partículas tienen interacción unas con otras.

El problema para configuración completa se aprecia en la **Figura 3.7**, esta configuración presenta 5 partículas que son $P1$, $P2$, $P3$, $P4$ y $P5$, además cada partícula está conectada con las otras cuatro, tomando como ejemplo la partícula $P2$ que tiene conexiones hacia $P1$, $P3$, $P4$ y $P5$. Al igual que en los problemas anteriores, cada partícula cuenta con su campo magnético externo asociado, para el caso de $P1$ su campo es 0.7, para $P2$ es de -0.8, $P3$ tiene 4.4, $P4$ tiene -1.3 y $P5$ tiene un campo de -2.1.

La tabla de posibles estados (dadas las combinaciones posibles) del sistema se muestra a continuación.

En la **Tabla 3.3** se muestran las combinaciones posibles para los estados del sistema, en este problema el estado de mínima energía es $|00000\rangle$ con costo de -10.9, este estado en particular es bastante interesante debido a que es el que "maximiza" (o minimiza) la ganancia (o pérdida) de energía con base en las interacciones o conexiones de las partículas.

Al igual que en los casos anteriores se puede establecer la ecuación que describe al operador de fase (o el Hamiltoniano del problema) de la siguiente forma.

P1	P2	P3	P4	P5	Costo
0	**0**	**0**	**0**	**0**	**-10.9**
0	0	0	0	1	-7.1
0	0	0	1	0	-5.5
0	0	0	1	1	-5.7
0	0	1	0	0	5.9
0	0	1	0	1	5.7
0	0	1	1	0	7.3
0	0	1	1	1	3.1
0	1	0	0	0	-4.5
0	1	0	0	1	-4.7
0	1	0	1	0	-3.1
0	1	0	1	1	-7.3
0	1	1	0	0	8.3
0	1	1	0	1	4.1
0	1	1	1	0	5.7
0	1	1	1	1	-2.5
1	0	0	0	0	-1.5
1	0	0	0	1	-1.7
1	0	0	1	0	-0.1
1	0	0	1	1	-4.3
1	0	1	0	0	11.3
1	0	1	0	1	7.1
1	0	1	1	0	8.7
1	0	1	1	1	0.5
1	1	0	0	0	0.9
1	1	0	0	1	-3.3
1	1	0	1	0	-1.7
1	1	0	1	1	-9.9
1	1	1	0	0	9.7
1	1	1	0	1	1.5
1	1	1	1	0	3.1
1	1	1	1	1	-9.1

Tabla 3.3: Combinaciones para problema ISM **completo** para 5 partículas.

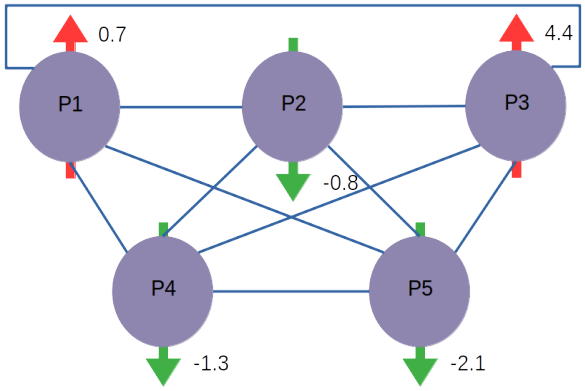


Figura 3.7: Problema **ISM** para un arreglo completo de 5 partículas.

$$U(C,\gamma) = (e^{i\gamma Z_1 Z_2} e^{i\gamma Z_1 Z_3} e^{i\gamma Z_1 Z_4} e^{i\gamma Z_1 Z_5} e^{i\gamma Z_2 Z_3} e^{i\gamma Z_2 Z_4} e^{i\gamma Z_2 Z_5} \dots$$
$$\dots e^{i\gamma Z_3 Z_4} e^{i\gamma Z_3 Z_5} e^{i\gamma Z_4 Z_5})(e^{i\gamma h_1 Z_1} e^{i\gamma h_2 Z_2} e^{i\gamma h_3 Z_3} e^{i\gamma h_4 Z_4} e^{i\gamma h_5 Z_5}) \tag{3.12}$$

Y para el operador mezclador se tiene la siguiente ecuación para describirlo.

$$U(C,\beta) = e^{i\beta X_1} e^{i\beta X_2} e^{i\beta X_3} e^{i\beta X_4} e^{i\beta X_5} \tag{3.13}$$

Como se puede observar en los distintos operadores que describen a cada problema de ISM vemos que realmente las ecuaciones son bastante similares, donde la diferencia entre una instancia de problema y otra es el número de interacciones (conexiones entre nodos) y el número de partículas, esto resulta bastante interesante porque plantea un esquema que permite abordar problemas de esta naturaleza (que están presentes en muchos sistemas físicos) con bastante facilidad, solo variando los parámetros y el número de partículas de cada problema.

3.2.4. Configuración lineal para Max-Cut

El problema de Max-Cut (como se explicó anteriormente) consiste en generar dos sub-grupos a partir de un grafo principal, y estos sub-grupos buscan maximizar el número de conexiones entre ellos. El tipo de problemas Max-Cut que se analizarán en este documento son los más simples, donde se tendrá un grafo que tendrá un cierto número de nodos y dicho grafo tendrá también conexiones entre nodos con configuración lineal, cíclica o completa, estas conexiones no tendrán peso asociado, y solo importará si existe o no una conexión entre los nodos.

Pasando a las instancias de problema para Max-Cut se inicia con la configuración lineal para el caso de 3 nodos.

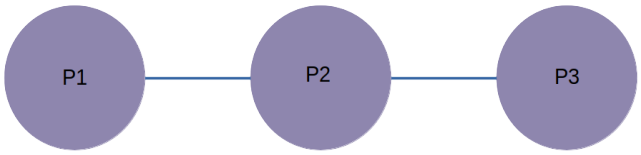


Figura 3.8: Problema **Max-Cut** para un arreglo lineal de 3 nodos.

P1	P2	P3	Conexiones
0	0	0	0
0	0	1	1
0	**1**	**0**	**2**
0	1	1	1
1	0	0	1
1	**0**	**1**	**2**
1	1	0	1
1	1	1	0

Tabla 3.4: Combinaciones para problema Max-Cut **lineal** de 3 nodos.

Observando la **Figura 3.8** se tiene el grafo para el problema lineal de 3 nodos, este problema es bastante sencillo en términos de Max-Cut, donde existen solo conexiones entre $P1$ a $P2$ y de $P2$ a $P3$.

Los resultados presentados en la **Tabla 3.4** se interpretan de la siguiente manera, cada nodo puede tener uno de dos estados posibles denotados como $|0\rangle$ o $|1\rangle$, el conjunto de nodos que se encuentran en el estado $|0\rangle$ se consideran parte de un sub-grupo (se podría interpretar como el subconjunto A mostrado anteriormente) del grafo del problema, y el conjunto de nodos que están en el estado $|1\rangle$ se encuentran en el otro sub-grupo (llámese subconjunto B mostrado anteriormente). Si los nodos vecinos se encuentran en subconjuntos opuestos esto aportará una conexión al total de conexiones, en caso contrario no se aporta conexión al total. Si tomamos como ejemplo las configuraciones óptimas para el problema Max-Cut lineal de 3 nodos presentes en la tabla anterior, para el primer caso (renglón cuatro de la tabla) se tiene un valor total de 2 conexiones que tiene asociado un estado $|\psi\rangle_1 = |010\rangle$, esto nos dice (debido a la instancia del problema) que los nodos $P1$ y $P3$ pertenecen al subconjunto A con estados $|0\rangle$ y que el nodo

$P2$ pertenece al subconjunto B con estados $|1\rangle$, esto se traduce en que $P1$ y $P2$ son vecinos y se encuentran en distintos subconjuntos esto aporta una unidad al valor total de conexiones, y dado que también $P2$ y $P3$ son vecinos y se encuentran en subconjuntos distintos esto aporta otra unidad al total, dando como resultado el valor de 2. La otra configuración que nos da el resultado máximo de conexiones es el inverso (o el negado) de $|010\rangle$ que se puede escribir como $|\psi\rangle_2 = NOT\,|010\rangle = |101\rangle$ que tiene un resultado igual a 2 en total de conexiones, en otras palabras, que el segundo estado óptimo corresponde al cambio de "pertenencia" de subconjunto entre $P1$ y $P3$ (como segundo subconjunto) y $P2$ (como primer subconjunto).

Siguiendo la secuencia de los problemas para ISM, se tiene la descripción del operador fase (el Hamiltoniano del problema) en la siguiente ecuación.

$$U(H_P) = U(C,\gamma) = e^{-i\gamma Z_1 Z_2} e^{-i\gamma Z_2 Z_3} \tag{3.14}$$

La **Ecuación 3.14** es bastante similar a las ecuaciones planteadas para los problemas de ISM, sin embargo, para Max-Cut no existe aporte externo (campo magnético externo) en el Hamiltoniano y este solo es influenciado por las conexiones entre los nodos. Otra diferencia que se observa es el signo de la unidad imaginaria, en este caso se maneja como $-i$ pero esto solo afecta a la fase global del sistema, y no al problema como tal, este signo se considera por convención al tratar este tipo de problemas con QAOA.

$$U(H_B) = U(C,\beta) = e^{-i\beta X_1} e^{-i\beta X_2} e^{-i\beta X_3} \tag{3.15}$$

El operador mezclador $U(H_B) = U(C,\beta)$ se describe en la **Ecuación 3.15**, y de igual forma que en el caso del operador de fase, es muy similar al de los problemas de ISM que solo tienen un signo de fase diferente presente en la unidad imaginaria $-i$.

3.2.5. Configuración cíclica para Max-Cut

De manera similar a lo planteado en el problema de ISM con configuración cíclica, en el problema para Max-Cut también se establecen 4 nodos que corresponden a $P1$, $P2$, $P3$ y $P4$.

Como se aprecia en la **Figura 3.9** la configuración del problema cíclico es muy similar al tipo de problemas de ISM, con la diferencia de que en este caso no existe el campo magnético externo, esto se podría relacionar con que solo las "conexiones" tienen aporte al Hamiltoniano del sistema.

En la **Tabla 3.5** se muestran las posibles combinaciones para los subconjuntos del grafo para el problema de 4 nodos en configuración cíclica, en este caso los estados que presentan el resultado óptimo son $|0101\rangle$ y su inverso $|1010\rangle$ con un total de conexiones de 4, el primer estado establece el primer subconjunto para

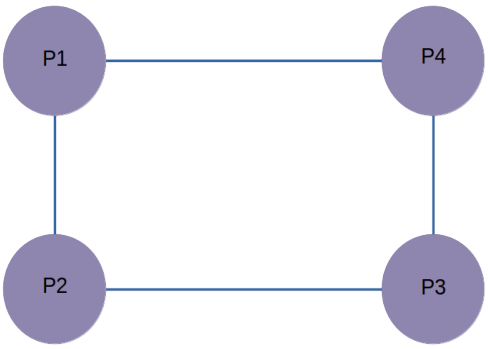


Figura 3.9: Problema **Max-Cut** para un arreglo cíclico de 4 nodos.

$P1$ y $P3$, y para el segundo subconjunto se tiene a $P2$ y $P4$ (y de forma inversa para el estado $|1010\rangle$).

Después de obtener los resultados posibles mostrados en la tabla anterior se puede transitar a establecer los modelos de los Hamiltonianos de problema y mezclador (operador de fase y operador mezclador).

$$U(H_P) = U(C,\gamma) = e^{-i\gamma Z_1 Z_2} e^{-i\gamma Z_2 Z_3} e^{-i\gamma Z_3 Z_4} e^{-i\gamma Z_1 Z_4} \tag{3.16}$$

El operador de fase para este problema se muestra en la **Ecuación 3.16**, este operador solo nos describe las conexiones existentes entre los diferentes nodos del grafo. Este operador muestra el camino que se tiene entre los nodos del grafo, es decir, como estos están conectados de tal manera que generan un ciclo usando 4 nodos.

$$U(H_B) = U(C,\beta) = e^{-i\beta X_1} e^{-i\beta X_2} e^{-i\beta X_3} e^{-i\beta X_4} \tag{3.17}$$

La **Ecuación 3.17** define al operador mezclador de este problema, la única diferencia que se tiene con el problema lineal es que en este problema se establecieron 4 nodos en el grafo.

3.2.6. Configuración completa para Max-Cut

Para la instancia de problema con configuración completa de Max-Cut se tiene un grafo de 5 nodos conectados de manera "completa", esto significa que cada nodo tiene una conexión directa con cualquier otro nodo del grafo.

P1	P2	P3	P4	Conexiones
0	0	0	0	0
0	0	0	1	2
0	0	1	0	2
0	0	1	1	2
0	1	0	0	2
0	**1**	**0**	**1**	**4**
0	1	1	0	2
0	1	1	1	2
1	0	0	0	2
1	0	0	1	2
1	**0**	**1**	**0**	**4**
1	0	1	1	2
1	1	0	0	2
1	1	0	1	2
1	1	1	0	2
1	1	1	1	0

Tabla 3.5: Combinaciones para problema Max-Cut **cíclico** de 4 nodos.

Para esta instancia se tiene un grafo de 5 nodos que se muestra en la **Figura 3.10**, esta configuración establece una conexión directa entre cada uno de los nodos $P1$, $P2$, $P3$, $P4$ y $P5$ del grafo.

Los datos generados para el caso completo de Max-Cut se pueden observar en la **Tabla 3.6**, los resultados obtenidos son bastante interesantes, ya que en este caso se tienen múltiples configuraciones que representan la configuración óptima del problema, el número de conexiones esperado para tener como óptimo es de 6, este resultado previo será importante para analizar en las simulaciones que se llevarán a cabo posteriormente en este texto y con ello observar tal vez otros comportamientos interesantes de este problema.

La forma del operador de fase para esta instancia se muestra en la siguiente ecuación.

$$\begin{aligned} U(H_P) = U(C,\gamma) = e^{-i\gamma Z_1 Z_2} e^{-i\gamma Z_1 Z_3} e^{-i\gamma Z_1 Z_4} e^{-i\gamma Z_1 Z_5} \dots \\ \dots e^{-i\gamma Z_2 Z_3} e^{-i\gamma Z_2 Z_4} e^{-i\gamma Z_2 Z_5} e^{-i\gamma Z_3 Z_4} e^{-i\gamma Z_3 Z_5} e^{-i\gamma Z_4 Z_5} \end{aligned} \tag{3.18}$$

De igual forma como en casos anteriores la **Ecuación 3.18** es muy similar a los problemas planteados en ISM, donde la única diferencia tangible es que la ecuación de arriba solo contempla las uniones entre nodos del grafo completo.

$$U(H_B) = U(C,\beta) = e^{-i\beta X_1} e^{-i\beta X_2} e^{-i\beta X_3} e^{-i\beta X_4} e^{-i\beta X_5} \tag{3.19}$$

P1	P2	P3	P4	P5	Conexiones
0	0	0	0	0	0
0	0	0	0	1	4
0	0	0	1	0	4
0	**0**	**0**	**1**	**1**	**6**
0	0	1	0	0	4
0	**0**	**1**	**0**	**1**	**6**
0	**0**	**1**	**1**	**0**	**6**
0	**0**	**1**	**1**	**1**	**6**
0	1	0	0	0	4
0	**1**	**0**	**0**	**1**	**6**
0	**1**	**0**	**1**	**0**	**6**
0	**1**	**0**	**1**	**1**	**6**
0	**1**	**1**	**0**	**0**	**6**
0	**1**	**1**	**0**	**1**	**6**
0	**1**	**1**	**1**	**0**	**6**
0	1	1	1	1	4
1	0	0	0	0	4
1	**0**	**0**	**0**	**1**	**6**
1	**0**	**0**	**1**	**0**	**6**
1	**0**	**0**	**1**	**1**	**6**
1	**0**	**1**	**0**	**0**	**6**
1	**0**	**1**	**0**	**1**	**6**
1	**0**	**1**	**1**	**0**	**6**
1	0	1	1	1	4
1	**1**	**0**	**0**	**0**	**6**
1	**1**	**0**	**0**	**1**	**6**
1	**1**	**0**	**1**	**0**	**6**
1	1	0	1	1	4
1	**1**	**1**	**0**	**0**	**6**
1	1	1	0	1	4
1	1	1	1	0	4
1	1	1	1	1	0

Tabla 3.6: Combinaciones para problema Max-Cut **completo** de 5 nodos.

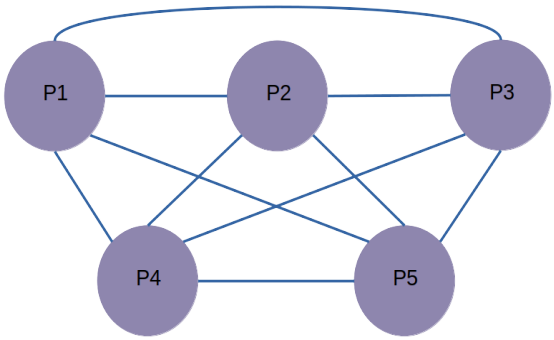


Figura 3.10: Problema **Max-Cut** para un arreglo completo de 5 nodos.

Y por último, en la **Ecuación 3.19** se tiene la forma del operador mezclador del problema con configuración completa a 5 nodos.

3.3. Descripción del método de optimización

QAOA permite obtener el estado óptimo para un problema dado, siempre y cuando los parámetros γ y β usados séanlos "ideales", estos parámetros pueden tratarse de múltiples γs y múltiples βs, si este es el caso, se tienen que encontrar múltiples parámetros ideales, sin embargo, para las instancias de problema descritas anteriormente solo existe un parámetro β y un parámetro γ. La etapa que se encarga de encontrar estos valores ideales para los parámetros es la etapa de optimización que se realiza por un algoritmo clásico dentro de QAOA, es por ello que a continuación se hablarán de los métodos que se buscan incorporar a QAOA con la finalidad de conocer las ventajas y desventajas (y con ello lograr una comparativa entre ellos) que estos presentan en problemas de tipo ISM y Max-Cut.

3.3.1. Método usando búsqueda exhaustiva

El primer método de optimización es llamado **Búsqueda Exhaustiva** (BE), este método es tal vez el más simple y conocido para resolver una gran cantidad de problemas, este método también es conocido como **Búsqueda de Fuerza Bruta** ya que la forma en como funciona es enumerando y probando cada una de las posibilidades de solución para un problema, función o condición a cumplir. [22]

Este método es considerado por la mayoría de los textos y autores como una

forma de búsqueda poco efectiva, ya que el algoritmo explora sobre todo el espacio de búsqueda la solución de un problema. El costo de usar este algoritmo para optimizar un problema puede ser muy elevado, pero también conlleva que este algoritmo pueda garantizar obtener el óptimo global del espacio de búsqueda, teniendo en cuenta que existe una limitante en precisión de la respuesta que se puede obtener, la búsqueda exhaustiva estará atada a la precisión que se desee para la respuesta. [22] [23]

Debido a que la limitante de obtener la solución óptima solo se encuentra en la precisión que se desea para la respuesta, la estrategia de BE es usada ampliamente como una primera aproximación al uso del algoritmo QAOA, para este trabajo si bien aplicar BE no representa una invocación, si presenta un estándar para probar otros métodos con respecto a la solución obtenida con el algoritmo búsqueda exhaustiva porque este método nos dará la solución (o soluciones) óptima con un rango de error (debido a la precisión disponible) y podrá ser comparada con los siguientes métodos de optimización para observar mejoras o carencias de otras técnicas. [33] [34]

Para la aplicación de BE en QAOA para las instancias de problema planteadas en este trabajo se considera de la siguiente manera:

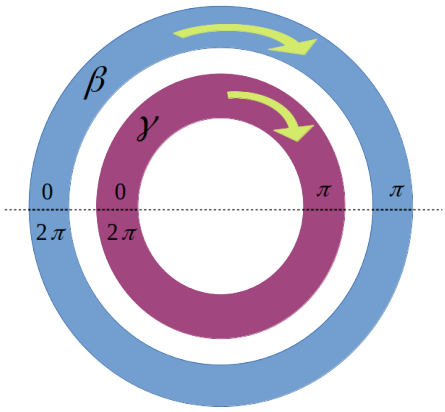


Figura 3.11: Parámetros γ y β para QAOA.

Lo que se muestra en la **Figura 3.11** es como se realiza el proceso de prueba de valores para los parámetros γ (operador de fase) y β (operador mezclador) usando BE, lo que se trata de representar en la figura es como existen dos parámetros independientes que "rotan", es decir, cambian de valores de forma independiente uno del otro. Los valores que pueden tomar ambos parámetros se encuentran contenidos entre $[0, 2\pi]$, este rango de valores si bien se considera en teoría como continuo, pero al implementarlo en un programa y en un circuito cuántico es necesario limitar el número de partes que se pueden adquirir de esta gama de valores, esto se debe principalmente a que en el circuito cuántico cada compuerta

debe ser generada para cumplir las necesidades de los valores elegidos para γ o β. Cuando el valor de β es elegido para un cierta iteración de BE se debe probar cada uno de los valores posibles para γ, este proceso se repite hasta que se prueba cada una de las combinaciones posibles, durante el proceso se van evaluando los resultados obtenidos con cada una de las combinaciones de parámetros y se va almacenando el mejor parámetro γ y el mejor parámetro β que al final de la aplicación de BE serán los parámetros que prevalecerán como respuesta final.

3.3.2. Método usando Búsqueda Local Iterativa

El siguiente método de optimización para obtener los parámetros de γ y β de QAOA es llamado **Búsqueda Local Iterativa** (BLI), este método es considerablemente más técnico y complejo que el caso de BE. El algoritmo BLI surge de la evolución del "Algoritmo de Ascenso en la Montaña Estocástico con Inicios Aleatorios" (AMEIA), la mejora se da con respecto a los inicios aleatorios, donde para el caso de AMEIA el inicio aleatorio puede darse en todo el espacio de búsqueda y en el caso de BLI existe un hiperparámetro que permite realizar un salto aleatorio con relación al mejor punto (o estado) alcanzado hasta el momento, en otras palabras, realiza una variación del último mejor punto (generar una área local) pero no es tan drástica como un inicio aleatorio en todo el espacio de búsqueda. [24]

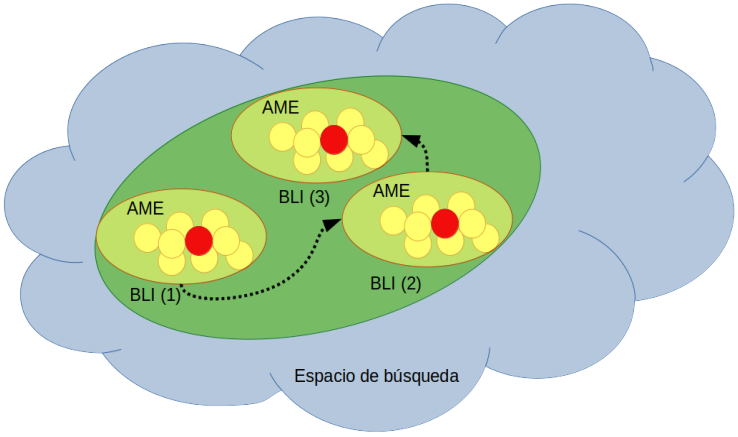


Figura 3.12: Representación ilustrativa para **Búsqueda Local Iterativa**.

En la **Figura 3.12** se busca ejemplificar el funcionamiento de BLI. Como primer paso el algoritmo genera un punto de inicio aleatorio en el espacio de búsqueda llámese "BLI (1)", este punto será el inicio de la búsqueda local, posteriormente se procede a implementar el AME (algoritmo de Ascenso en la Montaña Estocástico) que nos generará la búsqueda aleatoria en torno al punto elegido, después de encontrar nuestra mejor solución (actual) se genera un salto usando

el hiperparámetro de BLI y pasamos a "BLI (2)" que consiste en tomar como referencia la mejor solución actual y agregarle un factor de salto para transitar a otra zona relativamente cercana, este proceso se repite buscando cada vez una mejor solución a la anterior; si la solución nueva no es mejor a la anterior esta se descarta y el salto de BLI se realiza tomando en cuenta la mejor solución guardada.

Dependiendo del tipo de problema, BLI permite lograr encontrar el óptimo global del sistema teniendo en cuenta que se debe de proporcionar un número adecuando de iteraciones de búsqueda, esta solución óptima también dependerá del tipo de problema, ya que BLI al ser una heurística, no puede garantizar un funcionamiento perfecto para cada escenario.

Capítulo 4

Resultados y Análisis

En este capítulo se expondrán, analizarán y explicarán los resultados obtenidos para las distintas instancias de los problemas ISM y Max-Cut.

Los resultados obtenidos se separan en dos categorías, la primera categoría consiste en simulaciones de tipo **local**, simulaciones hechas en una computadora personal de baja capacidad, estos resultados normalmente se muestran como aproximaciones a los obtenidos en la realidad debido a la *simulación de comportamientos cuánticos* para asemejar al circuito cuántico que se esté simulando.

Por otro lado, la segunda categoría de simulaciones son realizadas de manera **real**, estos programas son enviados y ejecutados en la nube usando los servicios de IBM Quantum Computing los cuales brindan distintos tipos de computadoras cuánticas para ejecutar los circuitos cuánticos, estas computadoras pueden variar en calidad, capacidad y tecnología. Por el momento las simulaciones reales solo se pueden llevar a cabo en computadoras de máximo 5 qubits (donde actualmente IBM cuanta con computadoras de hasta 65 qubits), lo interesante de las simulaciones en hardware cuántico es observar que fenómenos se pueden extraer debido al estado actual de las computadoras cuánticas, y analizar si existen diferencias sustanciales entre las simulaciones locales y las reales. Estas simulaciones pueden también poseer un número elevado de errores dependiendo de la calibración y calidad del equipo cuántico que se esté usando para simular.

4.1. Simulaciones locales

El primer bloque de simulaciones se realizó usando una computadora personal de baja capacidad, debido a la baja complejidad de los problemas abordados la necesidad de un equipo más robusto no fue necesaria. Las características básicas del equipo son las siguientes:

- Sistema operativo (SO): Ubuntu 20.04.2 LTS.
- Tipo de SO: 64-Bits.

- Procesador: AMD A8-7410.
- Tarjeta gráfica: Radeon R5 Graphics.
- Memoria RAM: 4 GB.

Todos los programas fueron implementados usando Python 3, ya que los paquetes de **Qiskit** (paquetería para circuitos cuánticos de IBM) se encuentran disponibles solo para este lenguaje. Las características de la versión de Python usada, así como del paquete Qiskit se muestran a continuación.

- Lenguaje de programación: Python 3.8.10
- Versión de Qiskit: Qiskit 0.16.1

Especificaciones para simulaciones locales

Con estos aspectos mencionados ahora pasamos a hablar de las especificaciones que se aplicarán a las simulaciones locales. La lista de estas características se encuentran a continuación.

- Tipo de simulador: "qasm_simulator"
- Número de experimentos: 1000
- Resultado final: Mejor resultado guardado.
- Tipo de visualización: Histograma del mejor resultado guardado.

El primer elemento de la lista corresponde al tipo de simulador llamado **qasm_simulator** que es posiblemente una de las características más importantes, este tipo de simulador se encarga de ejecutar de manera "física" (para el caso local, simula el comportamiento de las compuertas y qubits del circuito) el circuito cuántico programado, es decir, que se generan todos los elementos que corresponden a los qubits, compuertas cuánticas y mediciones del circuito, esto se realiza tomando en cuenta el **número de experimentos**, esto hace que todo el experimento (todo el circuito cuántico) sea corrido 1000 veces y en cada simulación se generen datos a partir de las mediciones realizadas, estos datos son plasmados en el registro de bits (clásicos) del circuito y posterior a ello se realiza la estadística en relación con el número de experimentos realizados del circuito cuántico.

Es importante mencionar que existen otro tipo de simuladores disponibles en Qiskit, algunos de ellos incluyen: simulador de función de onda, simulador de aproximaciones, y uno que se usará más adelante llamado **proveedor cuántico**,

este último es el que se encarga del envío del programa al servicio de nube y que posteriormente sea implementado en la computadora cuántica elegida.

Algo más a tomar en cuenta para el método de Búsqueda Exhaustiva y Búsqueda Local Iterativa (o Iterada) es que la precisión dependerá del número de "pasos" (partes en las que se pueda dividir) que puedan tomarse dentro del rango $[0, 2\pi]$ que son los valores permitidos para γ y β. Las primeras simulaciones se realizarán con números distintos de divisiones con la finalidad de observar comportamientos no visibles con una sola simulación, además, dependiendo del tipo de problema se podrá proponer una modificación al algoritmo QAOA para observar su comportamiento.

Los valores que se obtendrán de los resultados serán:

- Costo total de apariciones (CTA).
- Estado obtenido (con mayor probabilidad).
- Número de apariciones (NA).
- Valores óptimos de γ y β.

El *costo total de apariciones* utiliza el costo de la evaluación (CE), que nos da el costo clásico de alguna combinación dada por el estado de los qubits, y esto se multiplica con el número de apariciones (NA) de la siguiente manera:

$$CTA = CE \times NA \tag{4.1}$$

El *número de apariciones* corresponde al número de ocurrencias de un estado en específico dado un experimento, esto se realiza por cada combinación de valores para γ y β. Debido a su naturaleza cuántica existe la posibilidad de que otros estados sean observados en las mediciones de cada experimento. QAOA busca generar la combinación de parámetros que permita tener con una mayor probabilidad de aparición al estado óptimo del sistema, visto de otra forma, elegir los parámetros que aumenten el número de apariciones del estado óptimo. El valor de CTA junto con la búsqueda del estado óptimo (dependiendo del problema) serán los factores que determinen la elección de una combinación sobre otra (para el primer método de evaluación).

El siguiente método para evaluar una cierta combinación de parámetros de los operadores de fase y mezclador es **Valor de Energía Esperado** (VEE), esta forma de evaluación es el estándar cuando se habla de QAOA.

$$Ve_i = \frac{CE_i \times NA_i}{NE} \tag{4.2}$$

Como puede verse, la **Ecuación 4.2** es muy similar al valor CTA, sin embargo, aquí aparece también el número de experimentos (NE), además, este valor se calcula para cada uno de los estados del sistema, los cuales se van sumando y el resultado final (en caso de que los parámetros de los operadores sean los correctos) nos dará el resultado óptimo (o un resultado muy cercano a este), este resultado denotado como VEE se tiene en la **Ecuación 4.3**.

$$VEE = \sum_{i=1}^{n} Ve_i \tag{4.3}$$

Este método es más útil en problemas reales debido a que no es necesario conocer el estado óptimo previamente a diferencia del método de evaluación anterior, la evaluación por búsqueda directa del estado óptimo y CTA solo será usado para el primer bloque de problemas de ISM usando BE, esto se realizará para identificar ventajas y desventajas de ambos métodos, pero sin perder de vista que son problemas bastante simples en los que se conocen muchas características verificables para comprobar el funcionamiento correcto de QAOA con los métodos de optimización.

4.1.1. Algoritmo QAOA usando Búsqueda Exhaustiva

El primer método de optimización simulado fue el de **Búsqueda Exhaustiva** (BE) para las dos clases de problemas que se abordarán: **ISM** y **Max-Cut**. El método de optimización por BE se ocupará en las tres instancias de problemas planteados (tanto para ISM como para Max-Cut) que son: configuración lineal de 3 partículas, configuración cíclica para 4 partículas y configuración completa para 5 partículas.

Simulaciones usando BE para ISM

El ISM es uno de los problemas más estudiados para implementar QAOA, y esto se debe a que es un problema que relaciona un modelo cuántico (aproximado usando Modelo de Ising) con un algoritmo cuántico como lo es QAOA, además, es bastante simple en su estructura porque solo existen dos elementos que aportan al operador de fase (Hamiltoniano del problema), estos elementos son las interacciones (o conexiones) entre partículas vecinas y el campo magnético externo asociado de forma individual a cada una de estas partículas.

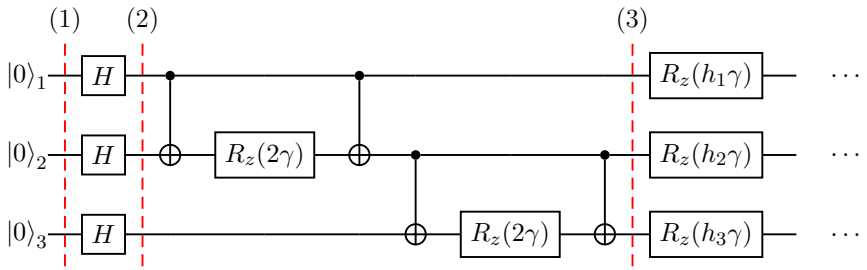
A continuación se estudiarán los problemas de 3 partículas en configuración lineal, 4 partículas en configuración cíclica, y 5 partículas en configuración completa. Estos problemas se simularán probando diferentes aspectos iniciando con el número de divisiones, que interviene directamente con la precisión de los elementos que pueden ser alcanzados en la gama de valores permitidos para los

parámetros γ y β que van de 0 a 2π, después se modifica el modelo base de QAOA aumentado a dos parámetros β para el caso de 4 y 5 partículas; y finalmente para el problema de 5 partículas se agregará también un nuevo parámetro γ teniendo en total cuatro parámetros llámese γ_1, γ_2, β_1 y β_2, esta es una técnica que se aplica para problemas con un mayor grado de complejidad, por ello es que en las primeras instancias no es necesario explorar esta vertiente para ser analizada.

También se analizará el efecto que tiene el tipo de método de evaluación, el primer método será la búsqueda directa del estado óptimo y el valor de CTA, y el segundo método será implementar el Valor de Energía Esperado (VEE). Estas pruebas solo se harán para esta primera etapa de las simulaciones y resultados, para los siguientes problemas como Max-Cut usando Búsqueda Exhaustiva, ISM usando Búsqueda Local Iterativa, etcétera; solo se hará uso del método por VEE, ya que es más practico en aplicaciones reales.

Simulación usando BE para configuración lineal de 3 partículas

Este problema se planteó como una configuración lineal de 3 partículas, estas partículas están etiquetadas como $P1$, $P2$ y $P3$ siendo $P2$ la partícula del centro que tiene conexiones con $P1$ y una conexión con $P3$. Además, los campos magnéticos individuales asociados para cada partícula son $h_1 = 0.7$ (para $P1$), $h_2 = 1.2$ ($P2$) y $h_3 = -1.6$ ($P3$).



En la **Figura 4.1** se observa el circuito cuántico que corresponde a la descripción de QAOA para el problema, la primera etapa (1) (las etapas se leen de punto de inicio a la derecha) es la inicialización de los estados a una superposición de los tres qubits (que representan a cada partícula del sistema), la segunda etapa (2) muestra la primera parte del operador de fase que identifica las conexiones que existen entre las partículas, la tercera etapa (3) es la segunda parte del operador de fase donde se adjuntan los campos de magnéticos externos por partícula, después se tiene la cuarta etapa (4) que es el operador mezclador (genera las rotaciones simples dado el parámetro β), y finalmente en la quinta etapa (5) se tienen los operadores de medición, estos operadores están conectados a los

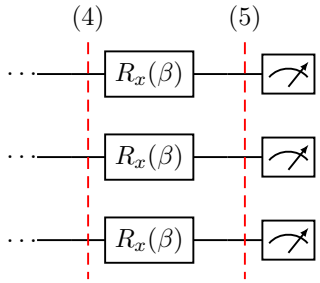


Figura 4.1: Circuito cuántico para el problema ISM de 3 partículas en configuración lineal.

Estado obtenido	$\|001\rangle$
Número de apariciones	771
CTA	-2698.5
Valor de γ	1.508802 π
Valor de β	1.508802 π

Tabla 4.1: Resultados de problema ISM con configuración lineal de 3 partículas con 25 divisiones.

registros clásicos que en este caso son de 3 bits (uno para cada qubit).

El orden de los estados generados de los qubits será de izquierda a derecha, siendo que el elemento más a la izquierda corresponderá a $P1$ y él más a la derecha será $P3$, por lo tanto, el elemento interno será $P2$.

En la **Tabla 4.1** se muestran los resultados obtenidos para el problema de 3 partículas en configuración lineal, estos resultados corresponden a 25 divisiones del rango de valores $[0, 2\pi]$. Como se tenía planteado desde la tabla de resultados del **Capítulo 3** el estado de mínima energía corresponde al estado $|001\rangle$, sin embargo, debido a que es un tipo de simulación local se esperaba que el número de apariciones fuese más alto, casi llegando a las 1000 apariciones, esto refleja que Qiskit tiene ya incorporados métodos para tratar de simular la aleatoriedad de las mediciones a un estado cuántico, las cuales debido a distintos factores no serán completamente correctas en muchos casos. En cuanto al costo total por apariciones o CTA es bastante normal dado el costo del estado óptimo y el número de veces que este ocurrió, este resultado nos sirve también de manera interna para buscar el estado de mínima energía tomando en cuenta el costo del estado que aparezca y el número de apariciones, esto hace que los estados que aporten menor CTA sean almacenados como mejores candidatos para estado óptimo. Finalmente, es también interesante ver que los parámetros γ y β tiene el mismo valor de 1.508802π (radianes), ya que la mayoría de las ocasiones este tipo de parámetros tienden a ser diferentes entre ellos.

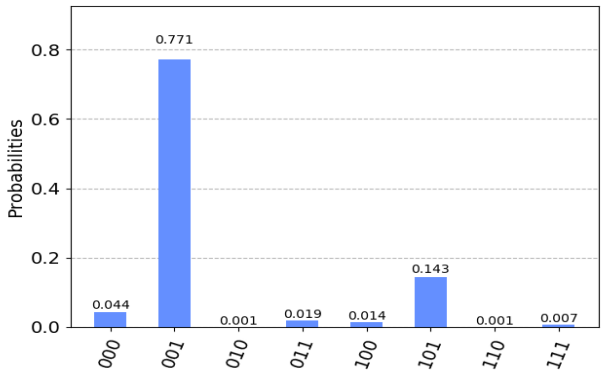


Figura 4.2: Histograma de mejor combinación de parámetros $\gamma = 1.508802\pi$ y $\beta = 1.508802\pi$ del problema de 3 partículas en configuración lineal.

Estado obtenido	$\|001\rangle$
VEE	-2.8496
Valor de γ	1.507964 π
Valor de β	1.507964 π

Tabla 4.2: Resultados de problema ISM con configuración lineal de 3 partículas con 25 divisiones usando VEE.

El número de apariciones por estado (del estado que presento el menor valor de CTA, la mejor combinación de parámetros) se grafican en el histograma de la **Figura 4.2**, esto concuerda con lo mencionado anteriormente donde a pesar de ser un simulador local que solo aproxima el comportamiento cuántico real, no se puede garantizar que la mejor combinación de parámetros sea efectiva para generar el estado de mínima energía en cada experimento, esto solo se podría si existiera una forma de generar una precisión infinitesimal para los parámetros γ y β, además, posiblemente incluso teniendo una precisión muy alta, debido al tipo de problema, podría ser necesario aumentar la profundidad de QAOA para mejorar la respuesta generada con relación al NA.

Implementando el método de **Valor de Energía Esperado** se obtuvieron los resultados que se observan en la **Tabla 4.2**, estos datos son muy similares a los obtenidos usando el método de evaluación con CTA.

Vemos que al igual que la tabla anterior, la **Figura 4.3** muestra prácticamente los mismos resultados que en el método de evaluación anterior.

Continuando con las simulaciones locales de este problema, ahora se plantea el uso de 50 fracciones para el rango de valores de γ y β.

Con la información obtenida de la **Tabla 4.3** vemos que el aumentar el número de divisiones ayudó a tener una mejor precisión en los parámetros, y con esto tener

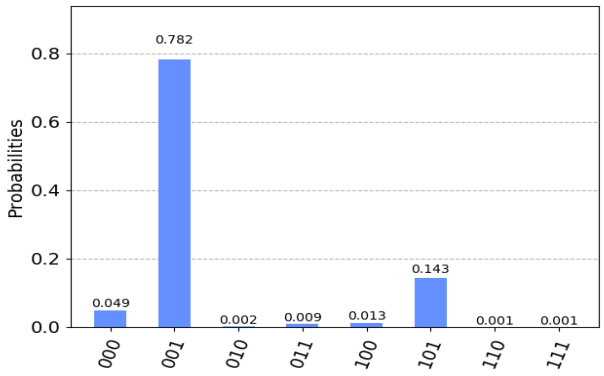


Figura 4.3: Histograma de mejor combinación de parámetros $\gamma = 1.507964\pi$ y $\beta = 1.507964\pi$ del problema de 3 partículas en configuración lineal usando VEE.

Estado obtenido	$\|001\rangle$
Número de apariciones	785
CTA	-2747.5
Valor de γ	1.382300 π
Valor de β	1.759291 π

Tabla 4.3: Resultados de problema ISM con configuración lineal de 3 partículas con 50 divisiones.

una mejor probabilidad de medir el estado óptimo usando QAOA, con un número de apariciones de 785 que es mayor al logrado por las 25 divisiones anteriores, y ahora también se observa que los valores de γ y β son distintos entre sí, γ teniendo un valor de 1.382300 y β un valor de 1.759291.

La representación de la mejor combinación de parámetros se muestra en la **Figura 4.4**, donde se ve el aumento del número de apariciones para el estado óptimo $|001\rangle$ y como los otros estados no deseados también disminuyeron en probabilidad de aparición.

De igual forma que en la simulación para 25 divisiones, se volvió a aplicar el método de evaluación con el VEE para el caso de 50 divisiones.

Estado óptimo	$\|001\rangle$
VEE	-2.8165
Valor de γ	1.507964 π
Valor de β	1.507964 π

Tabla 4.4: Resultados de problema ISM con configuración lineal de 3 partículas con 50 divisiones usando VEE.

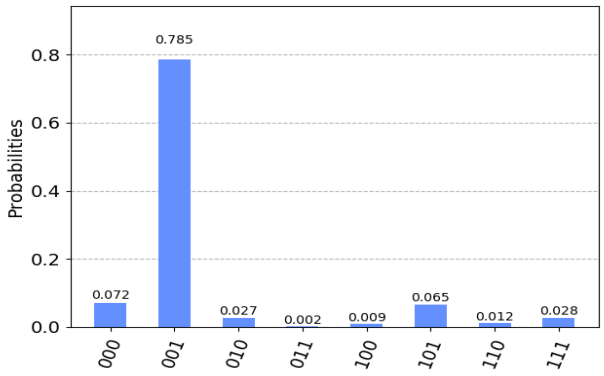


Figura 4.4: Histograma de mejor combinación de parámetros $\gamma = 1.382300\pi$ y $\beta = 1.759291\pi$ del problema de 3 partículas en configuración lineal.

Algo bastante interesante se tiene en los resultados de la **Tabla 4.4**, y es que a diferencia del método de CTA con búsqueda de estado óptimo, aquí la precisión del estado se mantuvo, e incluso los valores de γ y β permanecieron idénticos, caso contrario al método de evaluación anterior donde se presentó una diferencia entre dichos parámetros.

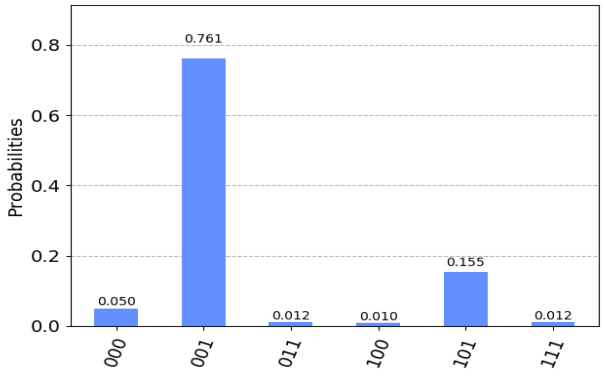


Figura 4.5: Histograma de mejor combinación de parámetros $\gamma = 1.507964\pi$ y $\beta = 1.507964\pi$ del problema de 3 partículas en configuración lineal usando VEE.

Los resultados para el caso de 100 divisiones se pueden consultar en el Apéndice, presentes en la **Tabla C.1** y en la **Figura C.1**.

Simulación usando BE para configuración cíclica de 4 partículas

Para el problema de ISM de configuración cíclica de 4 partículas primero se describe el circuito cuántico programado, este circuito incluye cuatro qubits que representan a cada partícula, además, a diferencia del circuito anterior en este caso existe una conexión entre la primera y última partícula, se debe mencionar que esta última interacción entre P_1 y P_4 puede expresarse de dos formas, usando la primera partícula como el "controlador" y la última como "objetivo" o de forma inversa, esto ocurre, ya que el proceso de entrelazamiento que se genera entre ambas puede verse en cualquier sentido. Las etapas que se explicaron para el circuito cuántico anterior funcionan también para este circuito, considerando solo los cambios mencionados.

Las simulaciones generadas usando 25 divisiones para el rango de parámetros de los operadores de fase y mezclador.

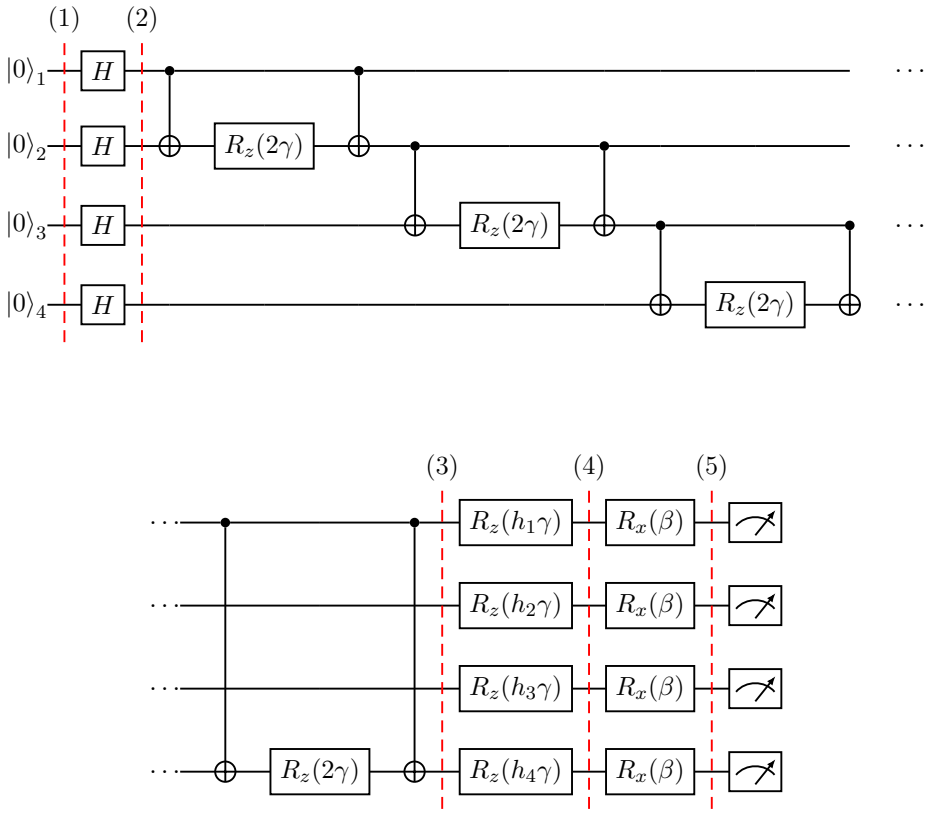


Figura 4.6: Circuito cuántico para el problema ISM de 4 partículas en configuración cíclica.

Analizando los datos obtenidos en la **Tabla 4.5** se tiene que el estado obtenido

Estado obtenido	$\|1101\rangle$
Número de apariciones	262
CTA	-1545.8
Valor de γ	1.256637 π
Valor de β	0.753982 π

Tabla 4.5: Resultados de problema ISM con configuración cíclica de 4 partículas con 25 divisiones.

coincide con la combinación predicha que proporciona el estado de mínima energía siendo $|1101\rangle$, sin embargo, aquí se tiene también que el número de apariciones (la probabilidad de observar el resultado midiendo QAOA) es bastante baja con 262 apariciones, esto representa solo $\frac{1}{4}$ del total de experimentos generados.

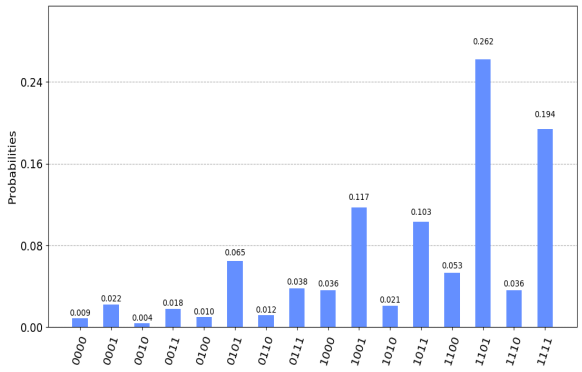


Figura 4.7: Histograma de mejor combinación de parámetros $\gamma = 1.256637\pi$ y $\beta = 0.753982\pi$ del problema de 4 partículas en configuración cíclica.

En la gráfica que se muestra en la **Figura 4.7** vemos con más detalle como afecta que el número de apariciones para el estado óptimo sea bajo, ya que esto propicia que la probabilidad de medir estados no óptimos también aumente como el caso del estado $|1111\rangle$, este fenómeno puede generarse debido a diferentes factores, pero normalmente se relaciona con una mala elección de parámetros de los operadores de fase y mezclador de QAOA o un modelo incorrecto de QAOA (falta de profundidad en los operadores).

Cambiando el método de evaluación a usar VEE, los resultados que se obtuvieron fueron los siguientes.

Ahora, los datos de la **Tabla 4.6** muestran resultados bastante interesantes, el primero es que el estado con mayor probabilidad de aparición es el estado $|0101\rangle$ que no es el estado de mínima energía del sistema, sin embargo se puede decir que está relativamente cerca de este, ya que su cálculo a la función de costo es de -4.9.

Estado obtenido	$\|0101\rangle$
VEE	-4.8044
Valor de γ	5.026548 π
Valor de β	1.759291 π

Tabla 4.6: Resultados de problema ISM con configuración cíclica de 4 partículas con 25 divisiones usando VEE.

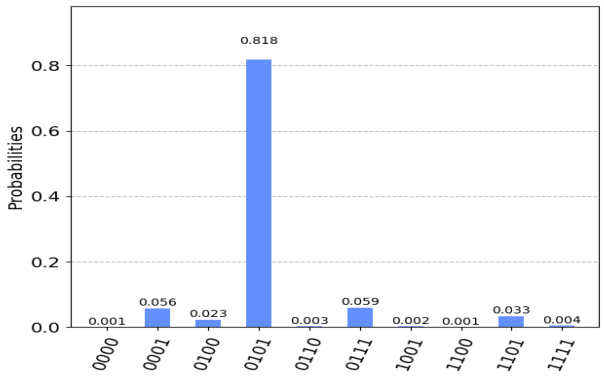


Figura 4.8: Histograma de mejor combinación de parámetros $\gamma = 5.026548\pi$ y $\beta = 1.759291\pi$ del problema de 4 partículas en configuración cíclica usando VEE.

El siguiente resultado interesante se aprecia en la **Figura 4.8**, y es que si bien no se obtuvo el estado óptimo con los parámetros elegidos, sí generó que existiera un estado predominante con poco más del 80 % de apariciones con respecto a los experimentos realizados, esto indica que posiblemente el problema se encuentre en que no se tenga una combinación adecuada de rotaciones (aplicación de operadores) que permita alcanzar el estado óptimo con mayor probabilidad, por ello ahora se continúa a aumentar la precisión (aumentar divisiones a 50) para analizar si existe una mejora o no en el resultado.

Con los resultados de la **Tabla 4.7** se puede inferir que aún aumentando el número de divisiones del rango $[0, 2\pi]$ de los parámetros no mejoró la respuesta, existe un aumento en el número de apariciones de 294 o un 30 % (con respecto al

Estado obtenido	$\|1101\rangle$
Número de apariciones	294
CTA	-1734.6
Valor de γ	1.256637 π
Valor de β	0.753982 π

Tabla 4.7: Resultados de problema ISM con configuración cíclica de 4 partículas con 50 divisiones.

Estado obtenido	$\|0101\rangle$
VEE	-4.8082
Valor de γ	4.900884 π
Valor de β	1.507964 π

Tabla 4.8: Resultados de problema ISM con configuración cíclica de 4 partículas con 50 divisiones usando VEE.

pasado de 262) y lo que conlleva un aumento en CTA, vemos que los valores de γ y β (con precisión de seis decimales) se mantienen iguales, con esto podemos esperar que aún aumentando las divisiones a 100 como en el problema anterior, no existirá una mejoría considerable en la respuesta.

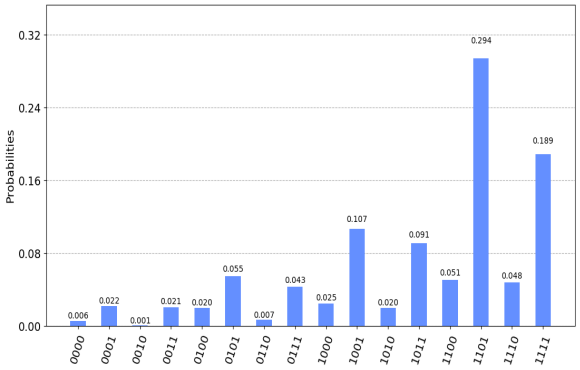


Figura 4.9: Histograma de mejor combinación de parámetros $\gamma = 1.256637\pi$ y $\beta = 0.753982\pi$ (50 divisiones) del problema de 4 partículas en configuración cíclica.

Agregando lo que muestra la **Figura 4.9** a los resultados de la tabla anterior, vemos que existe una pequeña mejora con respecto al resultado anterior de 25 divisiones, pero los valores de los operadores se mantienen prácticamente iguales.

Como podemos ver en la **Tabla 4.8** existe una mínima mejora en el valor de VEE aún relacionado al estado $|0101\rangle$ (estado no óptimo), y existe también una pequeña variación en el valor de γ y β comparado con el caso de 25 divisiones.

En la **Figura 4.10** se puede observar que la probabilidad de medición del estado $|0101\rangle$ decreció, y la probabilidad de aparición del estado $|0001\rangle$ aumentó a poco más de 20 %, este resultado es interesante porque el segundo mejor estado óptimo es el estado $|0001\rangle$ el cual no había tenido una buena probabilidad en la simulación para 25 divisiones, pero ahora vemos que existe un aumento de su aparición sobre otros estados.

Vemos que independientemente del método de evaluación implementado, existe una probabilidad muy baja de aparición del estado óptimo $|1101\rangle$, lo cual no

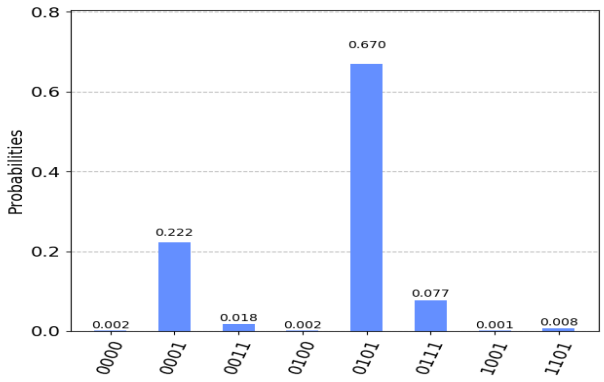


Figura 4.10: Histograma de mejor combinación de parámetros $\gamma = 4.900884\pi$ y $\beta = 1.507964\pi$ del problema de 4 partículas en configuración cíclica usando VEE.

es favorable para la correcta aplicación del algoritmo QAOA, ya que al menos (dependiendo de la aplicación) se busca tener una probabilidad de medición por encima del 70 % en el entendido que puede aumentarse esta precisión de manera "natural" (para el caso de Búsqueda Exhaustiva) usando un mayor número de divisiones si es que el algoritmo de QAOA está planteado de manera correcta para el problema en cuestión.

Estos resultados nos indican que tal vez el problema no pueda abordarse con solo una pareja de parámetros de γ y β. Para tratar estos inconvenientes al usar QAOA existen dos estrategias que son:

- Aumentar la complejidad del operador mezclador (agregar un nuevo hiperparámetro β_i).
- Agregar más parejas de valores γ_i y β_i.

El hecho de aumentar la complejidad (o el número) del operador mezclador se hace pensando en expandir el espacio de búsqueda del operador, ya que al tener solo un tipo de β se puede decir que solo existe una variable de rotación desde el punto de vista de la esfera de Bloch, entonces el agregar más variables de β permite al algoritmo explorar con mayor efectividad la superficie de la esfera, donde se encuentran todos los estados posibles para un qubit en particular. Para poder agregar este nuevo elemento de β_2 es necesario utilizar entrelazamiento entre el primer parámetro β_1, esto se realiza con miras a garantizar que pueda existir la generación de estados con **máximo entrelazamiento**, ya que sin agregar, por ejemplo, compuertas tipo $CNOT$ para lograr el entrelazamiento, podría darse el caso de que no podamos alcanzar los estados con entrelazamiento máximo debido a que las rotaciones que haría nuestro operador mezclador solo serían de forma individual y no de manera entrelazada.

Por otro lado, si el aumentar la complejidad del operador mezclador no es suficiente, se puede optar por aumentar las etapas de ambos operadores (fase y mezclador), esto se puede repetir las veces que sean necesarias a fin de poder garantizar que la exploración de estados cubra "completamente" el espacio de búsqueda. Este método es más demandante que el anterior porque al aumentar el número de parámetros también se aumenta la complejidad del algoritmo de optimización clásico necesario para encontrar los parámetros óptimos.

Aumento de complejidad de operador mezclador usando dos parámetros β_1 y β_2

Contemplando lo anterior, pasamos a el aumentó de complejidad del operador mezclador, esto se realizó agregando otro hiperparámetro β_2 con la misma gama de valores $[0, 2\pi]$, este parámetro se incluye al circuito usando entrelazamiento con compuertas tipo $CNOT$. El circuito cuántico resultante se muestra en la siguiente figura.

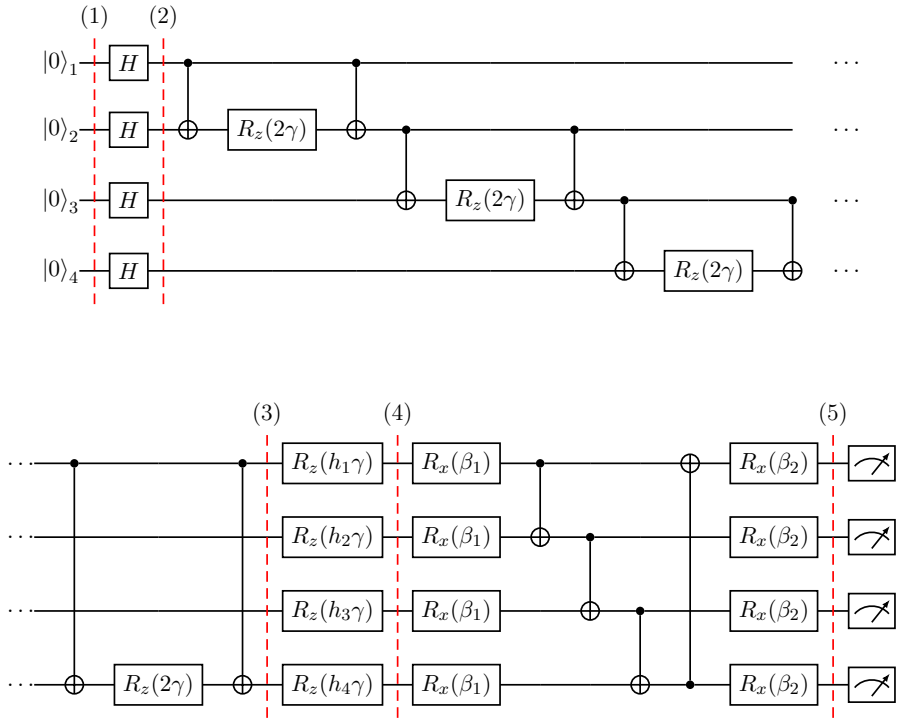


Figura 4.11: Circuito cuántico para el problema ISM de 4 partículas en configuración cíclica con dos parámetros β_1 y β_2.

En el circuito cuántico de la **Figura 4.11** se agrega la extensión del operador mezclador con el parámetro β_2 (que corresponde a la etapa (4)), aquí podemos ver

Estado obtenido	$\|1101\rangle$
Número de apariciones	479
CTA	-2826.1
Valor de γ	4.523893 π
Valor de β_1	1.507964 π
Valor de β_2	0.0 π

Tabla 4.9: Resultados de problema ISM con configuración cíclica de 4 partículas con 25 divisiones y dos parámetros β_1 y β_2.

como posterior al primer parámetro β_1 se ejecuta el entrelazamiento de los cuatro qubits y se aplica el segundo parámetro β_2 terminando el operador mezclador y pasando después a la etapa (5) con los operadores de medición.

Ya que el aumentar una nueva variable para realizar la Búsqueda Exhaustiva crea una mayor demanda de recursos de la computadora, el número de divisiones se planteó en 25 (para los tres hiperparámetros γ, β_1 y β_2) esto con miras en mantener un bajo aumento en la complejidad.

Con los datos que se observan en la **Tabla 4.9** vemos que existe una mejoría considerable respecto a los resultados anteriores, el número de apariciones ahora es de 479 con un valor de CTA -2826.1 y valores de los parámetros $\gamma = 4.523893\pi$, $\beta_1 = 1.507964\pi$ y $\beta_2 = 0.0\pi$. Estos resultados nos proporcionan dos aspectos importantes para mencionar, el primero es que la estrategia de aumentar la complejidad del operador de fase incorporando un nuevo parámetro β_2 entrelazado funciona de manera adecuada para aumentar la probabilidad de medición del estado objetivo, el otro aspecto se relaciona con el primero y es que debido a que el aumento del parámetro fue efectivo, esto nos indica que el estado óptimo buscado no se encontraba en el rango de búsqueda de una sola variable, no se encontraba en ese "grado de libertad" de la esfera de Bloch, por ello al agregar el nuevo parámetro es posible alcanzar el estado óptimo de manera natural y con esto se aumenta la probabilidad de aparición del mismo.

Incorporando la **Figura 4.12** a los resultados de la tabla vemos que el aumento de probabilidad de medición es cercano al 50 % y vemos también un aumento del estado $|0000\rangle$ (estado no óptimo), con esto podemos deducir que la estrategia de agregar un parámetro extra al operador mezclador aumentó la probabilidad de aparición del estado óptimo, pero aún no con una probabilidad aceptable.

Usando VEE como método de evaluación, se obtuvieron resultados bastante interesantes, la simulación se realizó usando 25 divisiones para el rango de parámetros de γ, β_1 y β_2.

Los resultados de la **Tabla 4.10** nos muestran que por primera vez usando VEE para el problema de 4 partículas cíclico se obtuvo el estado óptimo $|1101\rangle$ con mayor probabilidad, y este resultado es mejor que el alcanzado con el método

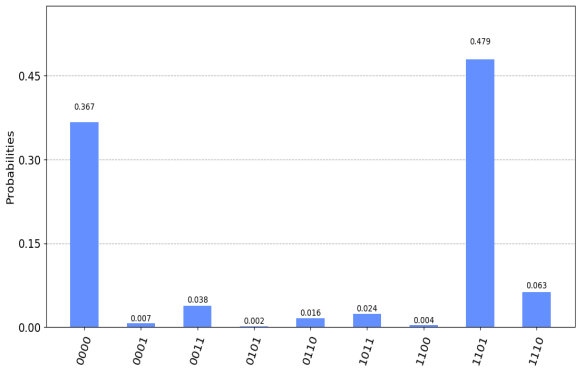


Figura 4.12: Histograma de mejor combinación de parámetros $\gamma = 3.839724\pi$, $\beta_1 = 4.886921\pi$ y $\beta_2 = 3.141592\pi$ (dos parámetros mezcladores) del problema de 4 partículas en configuración cíclica.

Estado obtenido	$\|1101\rangle$
VEE	-3.8814
Valor de γ	3.769911 π
Valor de β_1	4.775220 π
Valor de β_2	3.518583 π

Tabla 4.10: Resultados de problema ISM con configuración cíclica de 4 partículas con 25 divisiones y dos parámetros β_1 y β_2 usando VEE.

de evaluación usando CTA. Ya que si bien el valor esperado no esta "tan" cerca del costo del estado óptimo como en otros casos, se puede ver en la gráfica siguiente como la probabilidad de medición para $|1101\rangle$ es del 50 % prácticamente, y además, el siguiente estado más probable es el estado $|0001\rangle$ que es el segundo mejor estado del sistema.

En la **Figura 4.13** vemos que la probabilidad de aparición del estado óptimo es similar a la obtenida en el otro método de evaluación, sin embargo, el segundo mejor estado también está presente con 16 %, con esto podemos tener una probabilidad cercana al 70 % entre el estado óptimo y el segundo mejor estado.

En el Apéndice se puede encontrar los resultados para el método de evaluación usando CTA con 35 divisiones, el cual tuvo una pequeña mejora con un número de apariciones de 536, sin embargo, aún no fue suficiente para considerarlo una respuesta óptima. Los resultados se tienen en la **Tabla C.2** y en la gráfica de la **Figura C.2**.

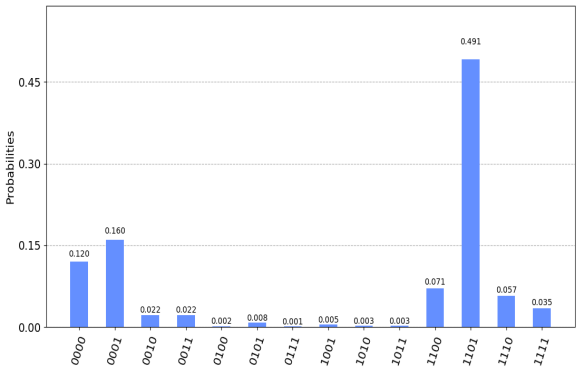
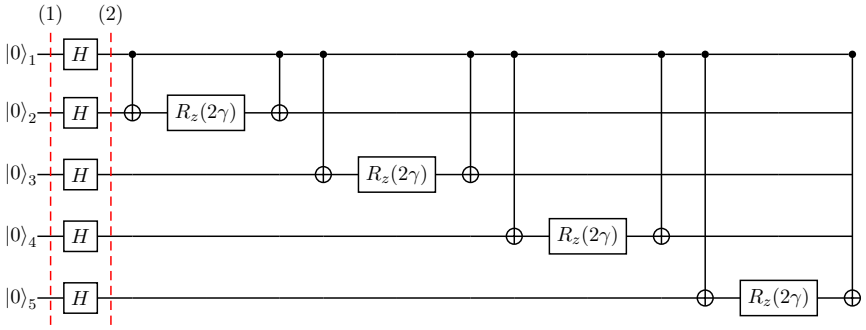


Figura 4.13: Histograma de mejor combinación de parámetros $\gamma = 3.769911\pi$, $\beta_1 = 4.775220\pi$ y $\beta_2 = 3.518583\pi$ (dos parámetros mezcladores) del problema de 4 partículas en configuración cíclica usando VEE.

Simulación usando BE para configuración completa de 5 partículas

La configuración completa usando 5 partículas es la más complicada en términos de interacciones entre partículas, ya que todas las partículas interactúan entre sí, lo cual genera un número elevado de conexiones comparado con las configuraciones pasadas. De forma similar al caso anterior se comprobará si el modelo simple (usando solo dos parámetros) de QAOA sirve para poder llegar a una solución aceptable del problema en términos del número de apariciones del estado óptimo, o si es necesario agregar uno o más parámetros al operador mezclador para mejorar la respuesta del algoritmo.



El circuito cuántico de la **Figura 4.14** muestra como se establece la configuración completa del problema, donde en este caso el principal factor es que cada partícula tiene relación con todas las otras. Todas las etapas son las mismas que se vieron en los casos anteriores en la versión más simple del algoritmo QAOA, en este circuito solo existe un par de parámetros llamados γ y β.

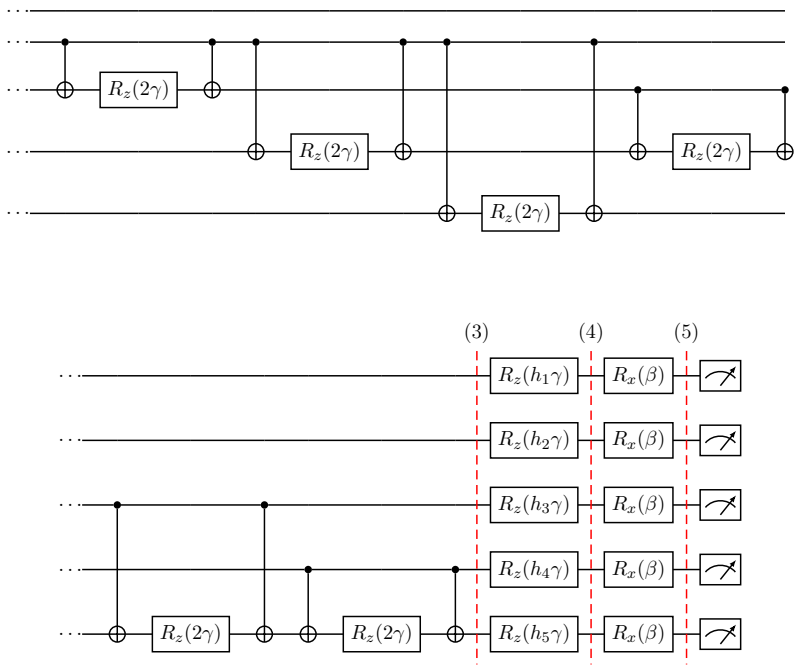


Figura 4.14: Circuito cuántico para el problema ISM de 5 partículas en configuración completa.

Iniciando con las simulaciones para el problema de 5 partículas se tienen los siguientes resultados generados usando 25 divisiones.

Con los datos de la **Tabla 4.11** vemos que a diferencia del caso anterior de 4 partículas, para este problema la probabilidad de aparición es casi del 50 % aún usando la versión más simple del algoritmo QAOA. El hecho de que el problema sea en principio más complejo que el anterior (relacionando el número de partículas y conexiones con la complejidad del problema); no es del todo correcto porque la complejidad del problema también se relaciona en gran medida con el estado óptimo que es necesario alcanzar, el estado óptimo para este problema es $|00000\rangle$ el cual no es un estado tan difícil de alcanzar con una correcta aplicación

Estado obtenido	$\|00000\rangle$
Número de apariciones	419
CTA	-4567.1
Valor de γ	5.026548 π
Valor de β	4.775220 π

Tabla 4.11: Resultados de problema ISM con configuración completa de 5 partículas con 25 divisiones.

Estado obtenido	$\|00000\rangle$
VEE	-7.9012
Valor de γ	5.026548 π
Valor de β	4.775220 π

Tabla 4.12: Resultados de problema ISM con configuración completa de 5 partículas con 25 divisiones usando VEE.

de compuertas, sin embargo, aún es necesario mejorar la respuesta con miras de obtener una mayor probabilidad de medición del estado óptimo.

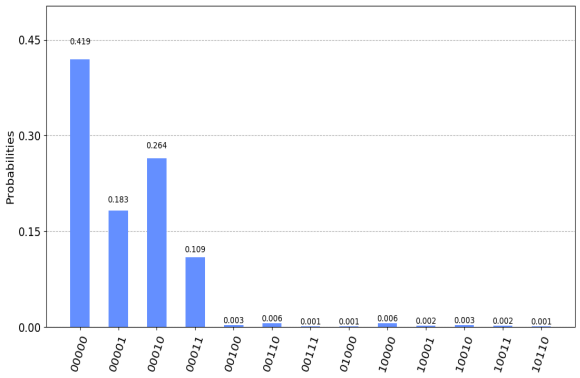


Figura 4.15: Histograma de mejor combinación de parámetros $\gamma = 5.026548\pi$ y $\beta = 4.775220\pi$ del problema de 5 partículas en configuración completa.

Viendo la **Figura 4.15** vemos que para la mejor combinación de parámetros γ y β el estado óptimo es el que se presenta con mayor probabilidad, pero existen también otros estados que aparecen con una probabilidad considerable que no son cercanos al estado óptimo, y se consideran estados no deseables.

Repitiendo el experimento anterior, pero ahora usado VEE como método de evaluación, se obtuvieron los siguientes resultados.

Analizando los resultados de la **Tabla 4.12** se puede observar que son muy similares a los obtenidos usando otro método de evaluación, la única diferencia es que en este caso VEE nos indica que el costo de evaluación del estado óptimo (considerando una ocurrencia del 100 %) aún no llega a ser -10.9 como se esperaba de la **Tabla 3.3**.

En el caso de la **Figura 4.16** vemos que los resultados son prácticamente idénticos al caso anterior, esto es de esperarse porque los valores de los parámetros son iguales para ambos métodos de evaluación.

Los resultados obtenidos para los casos de 50 divisiones usando el método de evaluación con CTA, que se pueden encontrar en la **Tabla C.3** y la **Figura**

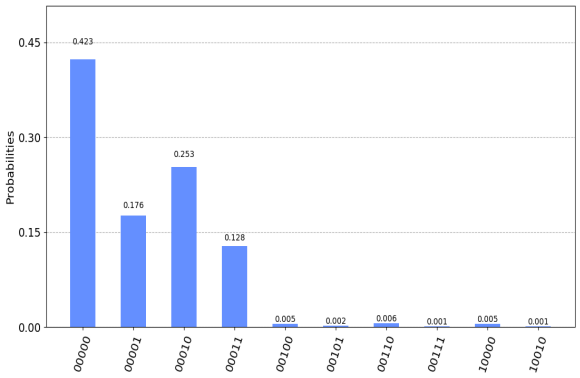


Figura 4.16: Histograma de mejor combinación de parámetros $\gamma = 5.026548\pi$ y $\beta = 4.775220\pi$ del problema de 5 partículas en configuración completa usando VEE.

Estado obtenido	$\|00000\rangle$
Número de apariciones	481
CTA	-5242.9
Valor de γ	5.780530 π
Valor de β_1	1.507964 π
Valor de β_2	3.267256 π

Tabla 4.13: Resultados de problema ISM con configuración completa de 5 partículas con 25 divisiones (dos parámetros β).

C.3; y para el caso de Valor de Energía Esperado (VEE) se encuentran en la **Tabla C.4** y en la **Figura C.4**. Estos resultados no fueron tan relevantes para discutirlos en esta sección.

Ya que los resultados en principio son adecuados dada la simplicidad del modelo usando en QAOA, pero que aún tienen una probabilidad baja para el estado óptimo (comparado con los casos anteriores), surge la pregunta al igual que en problema de 4 partículas, si existe alguna forma de mejorar la respuesta (aumentar probabilidad de aparición del estado óptimo) del algoritmo QAOA haciendo algunas modificaciones para aumentar su efectividad de exploración.

Aumento de complejidad de operador mezclador usando dos parámetros β_1 y β_2

La primera modificación del modelo de QAOA fue aumentar la complejidad del operador mezclador usando dos parámetros β.

Los resultados de la **Tabla 4.13** muestran que existe una pequeña mejoría

Estado óptimo	$\|00000\rangle$
VEE	-6.709200
Valor de γ	5.026548 π
Valor de β_1	4.523893 π
Valor de β_2	0.0 π

Tabla 4.14: Resultados de problema ISM con configuración completa de 5 partículas con 25 divisiones (dos parámetros β) usando VEE.

con respecto al caso anterior de 25 divisiones, ya que el número de apariciones es casi del 50 %, así mismo el valor de CTA mejoró con respecto a la simulación con solo dos parámetros del algoritmo QAOA.

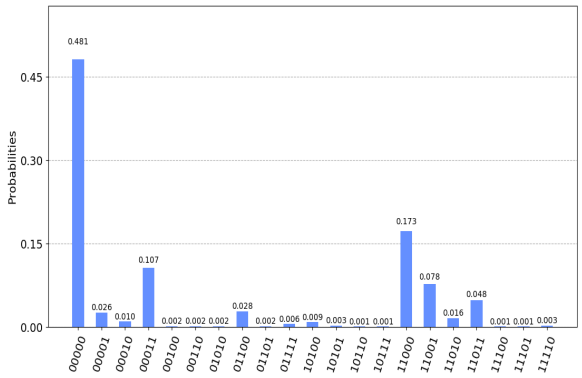


Figura 4.17: Histograma de mejor combinación de parámetros $\gamma = 5.780530\pi$, $\beta_1 = 1.507964\pi$ y $\beta_2 = 3.267256\pi$ del problema de 5 partículas en configuración completa.

En el histograma que se muestra en la **Figura 4.17** se aprecia una mejoría, ya que el estado óptimo se muestra con una mayor probabilidad que cualquier otro estado, pero aún no se tiene una probabilidad cercana al 70 % que sería considerado como aceptable.

Usando el método de evaluación con VEE y aplicando el segundo parámetro de β_2 con 25 divisiones se extrajeron los siguientes resultados presentes en la **Tabla 4.14**, la tabla nos muestra que los resultados correspondientes al VEE fueron peores a los obtenidos con el modelo simple de QAOA, esto es interesante, ya que se esperaba que al aumentar a dos parámetros en el operador mezclador existiera una mejor posibilidad de alcanzar el estado óptimo, sin embargo, esto puede indicar que el problema no pueda ser tratado solamente como un operador mezclador complejo.

Agregando la información estadística de la **Figura 4.18** se tiene que definitivamente no existe una mejora notable en cuanto a la probabilidad de medición

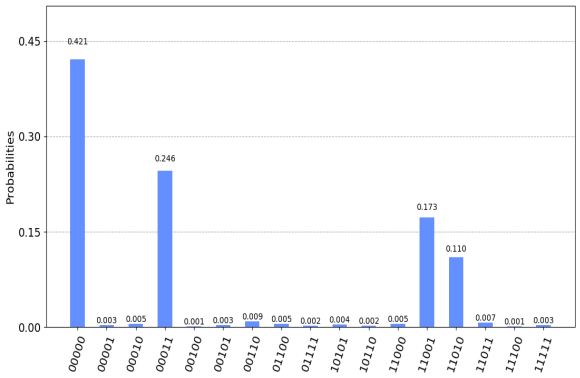


Figura 4.18: Histograma de mejor combinación de parámetros $\gamma = 5.026548\pi$, $\beta_1 = 4.523893\pi$ y $\beta_2 = 0.0\pi$ del problema de 5 partículas en configuración completa usando VEE.

del estado óptimo se refiere, esto es distinto a lo encontrado con el otro método de evaluación.

Aumento a dos parejas de parámetros γ_1 y γ_2, con β_1 y β_2

Los resultados anteriores (usando el modelo simple y modificado de QAOA con dos βs) sugieren que posiblemente se necesite aumentar la complejidad del algoritmo QAOA, esto se logra mediante dos estrategias, la primera es agregando más parámetros al operador mezclador; que corresponde a la estrategia ya simulada en estos resultados, y la segunda manera es aumentando la aplicación de operadores fase y mezcladores de manera consecutiva, como se muestra en la siguiente ecuación.

$$|\psi_{\gamma\beta}\rangle = U(B,\beta_p)U(C,\gamma_p)\dots U(B,\beta_2)U(C,\gamma_2)U(B,\beta_1)U(C,\gamma_1)\,|s\rangle \qquad (4.4)$$

El segundo método no es tan recomendable como el primero ya que incrementa el número de parámetros en proporción $2p$ donde p es el "nivel" o profundidad del algoritmo QAOA (cuantos pares de parámetros γ y β), y esto se resume en un aumento en la cantidad de parámetros a optimizar. Con lo anterior mencionado, se puede experimentar con un incremento a 2 parámetros γ y 2 parámetros β para ambos métodos de evaluación.

Usando el método de evaluación con CTA se tienen los datos de la **Tabla 4.15** que nos muestran como existe una pequeña mejoría con respecto a los casos pasados teniendo poco más del 50 % de probabilidad de medición.

Estado obtenido	$\|00000\rangle$
Número de apariciones	506
CTA	-5515.4
Valor de γ_1	5.780530 π
Valor de γ_2	2.010619 π
Valor de β_1	1.507964 π
Valor de β_2	3.015928 π

Tabla 4.15: Resultados de problema ISM con configuración completa de 5 partículas con 25 divisiones con dos parámetros γ y dos parámetros β.

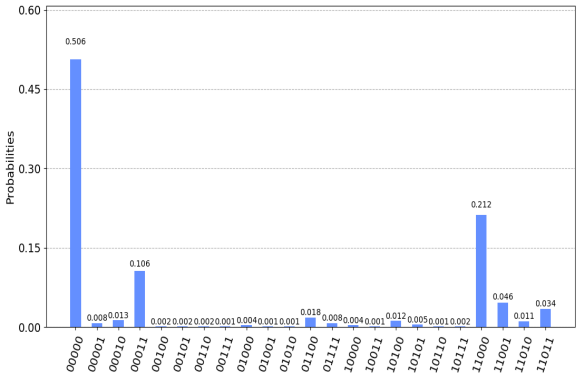


Figura 4.19: Histograma de mejor combinación de parámetros $\gamma_1 = 5.780530\pi$, $\gamma_2 = 2.010619\pi$, $\beta_1 = 1.507964\pi$ y $\beta_2 = 3.015928\pi$ del problema de 5 partículas en configuración completa.

Y agregando la representación visual de la **Figura 4.19** vemos como la probabilidad de aparición del estado óptimo aumento en comparación a las otras simulaciones, además se tiene una disminución de las probabilidades para los estados no deseados.

Pasando al método de evaluación por VEE se tienen los resultados de la **Tabla 4.16** donde vemos que el estado con mayor probabilidad de aparición es el estado $|00000\rangle$ que es el estado óptimo del sistema, comparado con el modelo anterior de QAOA con solo dos βs, vemos que existe una mejoría bastante significativa en el valor VEE.

El histograma de la **Figura 4.20** muestra al estado óptimo con mayor probabilidad de aparición, sin embargo, aún existen estados no deseados como $|00010\rangle$ y $|00011\rangle$ que no tienen un costo de evaluación cercano al óptimo, esto podría indicar que se necesite mejorar la precisión de la respuesta agregando más divisiones al rango de valores de los parámetros, o inclusive que sea necesario agregar un parámetro extra para el operador mezclador y combinar ambas estrategias

Estado obtenido	$\|00000\rangle$
VEE	-8.1286
Valor de γ_1	0.0 π
Valor de γ_2	5.026548 π
Valor de β_1	1.005309 π
Valor de β_2	4.775220 π

Tabla 4.16: Resultados de problema ISM con configuración completa de 5 partículas con 25 divisiones con dos parámetros γ y dos parámetros β usando VEE.

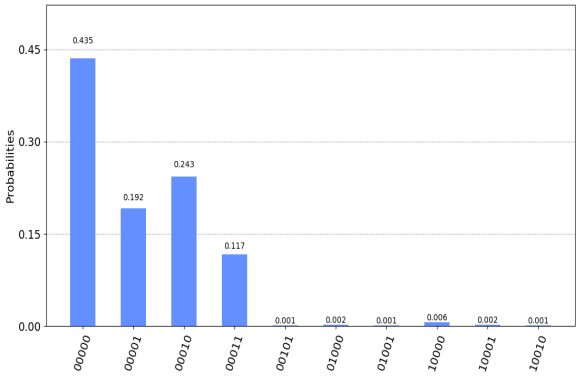


Figura 4.20: Histograma de mejor combinación de parámetros $\gamma_1 = 1.507964\pi$, $\gamma_2 = 3.015928\pi$, $\beta_1 = 4.523893\pi$ y $\beta_2 = 5.780530\pi$ del problema de 5 partículas en configuración completa usando VEE.

en una; pero debido a la demanda que estos experimentos representan no se abordarán en este trabajo.

Es interesante que para el caso del problema de ISM de configuración **completa**, el estado con mínima energía es una de las dos configuraciones posibles que maximizan (o minimizan) la perdida (o ganancia) de energía debido a las interacciones (conexiones) entre partículas, y esto lo podemos relacionar directamente con tipo de configuración completa, es decir, un grafo completo; donde las interacciones se presentan en mayor número que en los casos lineales y cíclicos, esto conlleva a que el estado tienda a generar una menor energía (que es el objetivo) en configuraciones donde las conexiones entre partículas aporten una mayor perdida.

Simulaciones usando BE para Max-Cut

La siguiente clase de problemas para simular son los de tipo **Max-Cut**, como se mencionó, este tipo de problemas buscan formar dos subconjuntos a partir de

un grafo principal donde los subconjuntos tengan el máximo número de conexiones entre ellos. Esta clase de problemas es similar a la anterior (en planteamiento) con la diferencia que no existe un campo magnético externo y solo existe un aporte de la conexión entre nodos que influye en el operador de fase.

Simulaciones usando BE para configuración lineal de 3 nodos

Iniciando con las simulaciones de Max-Cut se tiene el problema de 3 nodos en configuración lineal, el diagrama de circuito cuántico que representa este problema es muy similar al de ISM, pero tiene la diferencia de que no existen las compuertas $R_z(h_i\gamma)$.

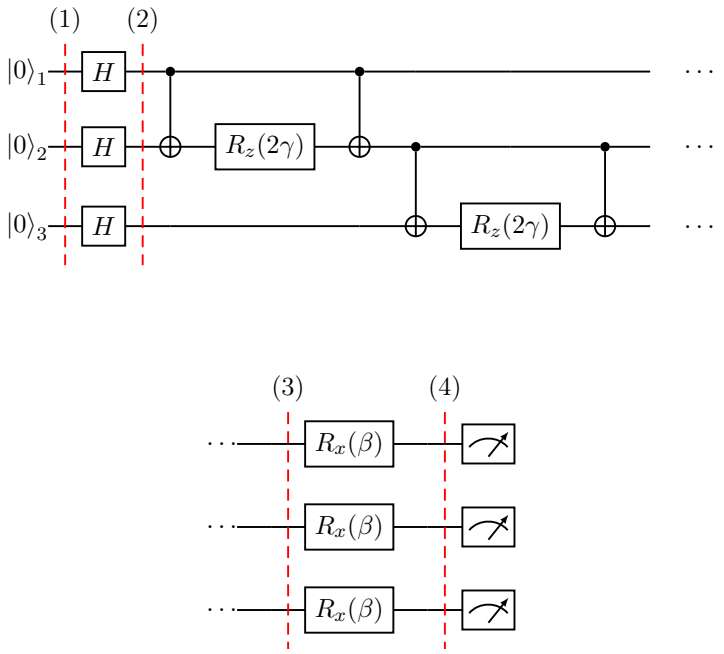


Figura 4.21: Circuito cuántico para el problema **Max-Cut** de 3 nodos en configuración lineal.

Revisando el circuito de la **Figura 4.21** vemos como solo se pueden identificar cuatro etapas, que inician con la sección de la etapa (1) (léase de número de etapa hacia la derecha) que se tiene la aplicación de las compuertas Hadamard H que se encargan de generar la superposición de los estados base para los qubits del circuito; la etapa (2) es el operador de fase con el parámetro γ, el cual solo contiene las conexiones entre nodos; la etapa (3) contiene al operador mezclador con el parámetro β, y por último se tiene la etapa (4) que corresponde a los operadores de medición, estos operadores al igual que en los circuitos anteriores,

Estado obtenido	$\|101\rangle$
VEE	1.658
Valor de γ	0.502654 π
Valor de β	2.261946 π

Tabla 4.17: Resultados de problema Max-Cut con configuración lineal de 3 nodos con 25 divisiones.

conectan los observables (posterior a su medición) del registro cuántico al registro clásico para poder ser interpretados por la computadora.

Para la primera simulación aplicando BE con 25 divisiones, los resultados generados son los siguientes.

Los datos que contiene la **Tabla 4.17** muestran que el valor esperado $VEE = 1.658$ se encuentra relativamente cerca del valor óptimo de 2, esto se puede relacionar a que existen dos estados óptimos que son $|010\rangle$ y $|101\rangle$, y además los estados no deseados tienen un valor cercano a estos lo que hace más difícil generar una "separación" aceptable entre estos.

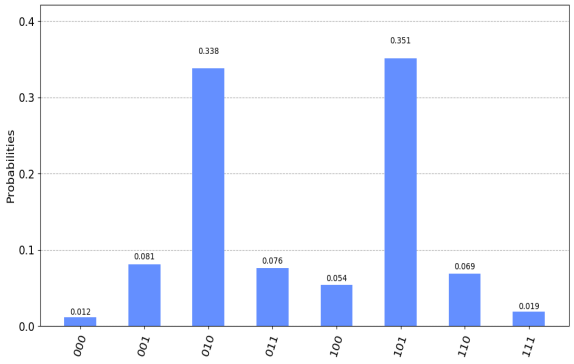


Figura 4.22: Histograma de mejor combinación de parámetros $\gamma = 0.502654\pi$ y $\beta = 2.261946\pi$ del problema de 3 nodos en configuración lineal.

La **Figura 4.22** muestra los resultados para la mejor combinación de parámetros, aquí se puede ver que los estados óptimos $|010\rangle$ y $|101\rangle$ tienen casi el 70 % de probabilidad de aparición de los 1000 experimentos realizados. También se aprecia algo interesante con los estados no deseados, y es que los estados $|000\rangle$ y $|111\rangle$ son los que tiene las probabilidades más bajas de aparición debido a que su costo de evaluación es el menor con 0 y el resto de los estados (exceptuando los óptimos) tienen relativamente la misma probabilidad de medición dado su valor de costo que es de 1 (entiéndase que el "costo" hace referencia al número de conexiones entre los subconjuntos generados).

Después al aumentar el número de divisiones a 50 no se encontró una mejoría

significativa, por ello los resultados son ingresados al Apéndice 2. Estos resultados se pueden comprobar en la **Tabla C.5** y en la **Figura C.5**.

Es importante recordar que esta subsección y las siguientes solo usarán el método de evaluación por VEE, esto se hará para mantener un tamaño razonable en la sección de resultados y debido a que VEE es el método de evaluación estándar en QAOA (véase la introducción a la sección de **Resultados**).

Simulaciones usando BE para configuración cíclica de 4 nodos

Pasando a la siguiente instancia, tenemos el problema de configuración cíclica de Max-Cut para 4 nodos, el circuito cuántico para este problema es el siguiente.

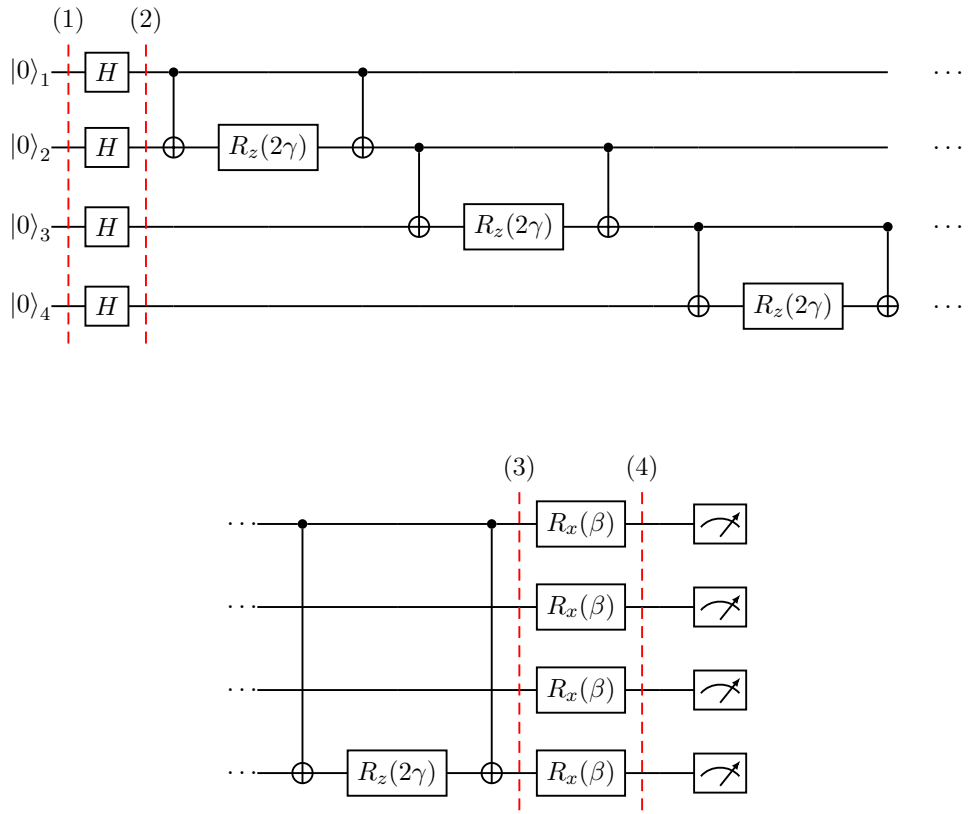


Figura 4.23: Circuito cuántico para el problema **Max-Cut** de 4 nodos en configuración cíclica.

Las primeras simulaciones generadas para este problema son usando 25 divisiones para el rango de valores de los parámetros de los operadores de fase y mezclador, los resultados se muestran a continuación.

Estado obtenido	$\|1010\rangle$
VEE	2.08800
Valor de γ	1.507964 π
Valor de β	2.764601 π

Tabla 4.18: Resultados de problema Max-Cut con configuración cíclica de 4 nodos con 25 divisiones.

Los datos de la **Tabla 4.18** muestran que el VEE está muy alejado del valor óptimo que es 4, esto a pesar de que el estado con mayor número de apariciones fue uno de los dos estados óptimos, estos resultados pueden analizarse de mejor forma con el histograma siguiente.

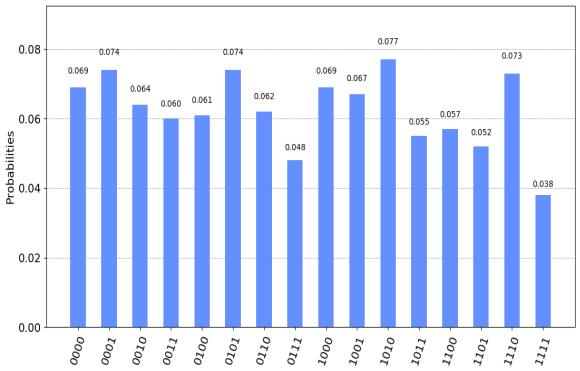


Figura 4.24: Histograma de mejor combinación de parámetros $\gamma = 1.507964\pi$ y $\beta = 2.764601\pi$ del problema de 4 partículas en configuración cíclica.

Observando la **Figura 4.24** vemos que todos los estados tienen casi la misma probabilidad de medición, esto indica que nuestro modelo no es adecuado para atacar el problema planteado, además al no tener un estado predominante (un estado con mayor probabilidad de aparición sobre el resto de estados) puede indicar que el problema esté en el operador mezclador y aunque busquemos aumentar el número de divisiones podríamos no tener un mejor resultado. Ahora se procede a aumentar a dos parámetros el operador mezclador de forma similar al caso de ISM de 4 partículas.

Aumento de complejidad de operador mezclador usando dos parámetros β_1 y β_2

Al igual que en el experimento de ISM si la respuesta del modelo simple (solo un par de parámetros) no es adecuada, el primer paso es optar por aumentar la complejidad del operador mezclador, esto se hace debido a que el problema

Estado obtenido	$\|1010\rangle$
VEE	2.618
Valor de γ	5.780530 π
Valor de β_1	4.775220 π
Valor de β_2	1.507964 π

Tabla 4.19: Resultados de problema Max-Cut con configuración cíclica de 4 nodos con 25 divisiones y dos parámetros β_1 y β_2.

puede recaer en que el estado óptimo del sistema no se pueda alcanzar realizando rotaciones en un solo eje, entonces debido a que el operador de fase "prepara" los qubits en lo que se refiere al Hamiltoniano del problema, y el operador encargado de establecer un estado de forma predominante es el operador mezclador, por ello aumentar las variables en las que se puede rotar el estado de QAOA mejora la respuesta final del algoritmo.

Al aumentar tan solo una variable cuando se habla de BE es sinónimo de incrementar el tiempo de ejecución y la cantidad de recursos que se necesitan para optimizar dichas variables, por ello las simulaciones realizadas siempre son con una cantidad reducida de divisiones para mantener este aumento dentro de los márgenes computables del dispositivo.

El diagrama de circuito es muy similar a los anteriores con la diferencia del aumento del parámetro β_2 de manera similar al caso del circuito de la **Figura 4.11**.

Las simulaciones generadas para este operador mezclador con dos variables se hicieron con 25 divisiones para el rango de valores de los parámetros.

La **Tabla 4.19** nos enseña que el valor VEE tuvo un aumento a 2.618 lo cual es bastante positivo, pero aún está muy lejos del valor evaluado para los estados óptimos.

En la **Figura 4.25** vemos que efectivamente existe un aumento en la probabilidad de aparición de los estados óptimos, pero aun sumando ambas probabilidades no llegamos ni al 50 %, lo cual indica que también este modelo pueda no ser suficiente para abordar el problema y obtener una solución óptima con una probabilidad suficiente.

Aumento a dos parejas de parámetros γ_1 y γ_2, con β_1 y β_2

Esta estrategia es muy similar a la empleada para el caso de 5 partículas de ISM, pero ahora se busca aplicarla para el problema de 4 nodos en configuración cíclica, con miras a mejorar la solución generada por los otros modelos.

Ahora vemos como el valor esperado VEE de la **Figura 4.20** es prácticamente el valor de costo evaluado para cualquiera de los dos estados óptimos, además, el

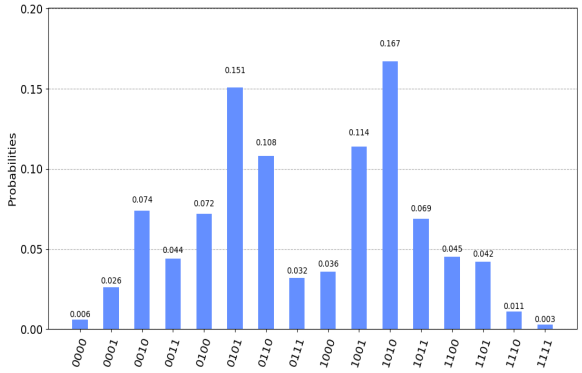


Figura 4.25: Histograma de mejor combinación de parámetros $\gamma = 5.780530\pi$, $\beta_1 = 4.775220\pi$ y $\beta_2 = 1.507964\pi$ del problema de 4 nodos en configuración cíclica.

Estado obtenido	$\|0101\rangle$
VEE	3.9819
Valor de γ_1	1.507964 π
Valor de γ_2	0.753982 π
Valor de β_1	4.775220 π
Valor de β_2	4.775220 π

Tabla 4.20: Resultados de problema Max-Cut con configuración cíclica de 4 nodos con 25 divisiones y dos parámetros γ_1, γ_2, y dos parámetros β_1 y β_2.

estado con mayor apariciones es uno de los estados óptimos que es $|0101\rangle$.

La gráfica que vemos en la **Figura 4.26** muestra claramente una tendencia de más del 95 % de probabilidad de medición sobre el estado óptimo $|0101\rangle$ lo cual es (hasta el momento) la mejor respuesta generada de todas las simulaciones e instancias de problema investigadas. Esto nos indica que el aumentar el número de operadores con sus respectivos parámetros fue la mejor estrategia para garantizar la prevalencia del estado óptimo, y a su vez decrementar la aparición de los estados no deseados.

Simulaciones usando BE para configuración completa de 5 nodos

Pasando al problema de 5 nodos en configuración completa se tiene que el circuito cuántico que lo representa es bastante similar al usado en el caso de 5 partículas de ISM, la única modificación que se tiene con respecto a ese modelo es que no existe la etapa de las compuertas $R_z(h_i\gamma)$ (véase la **Figura 4.14** entre etapa (3) y (4)), debido a esta similitud no es necesario presentar el diagrama

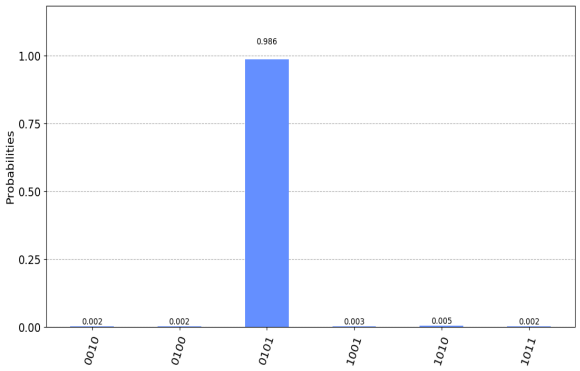


Figura 4.26: Histograma de mejor combinación de parámetros $\gamma_1 = 1.507964\pi$, $\gamma_2 = 0.753982\pi$, $\beta_1 = 4.775220\pi$ y $\beta_2 = 4.775220\pi$ del problema de 4 nodos en configuración cíclica.

Estado obtenido	$\|01010\rangle$
VEE	3.514
Valor de γ	1.256637 π
Valor de β	4.775220 π

Tabla 4.21: Resultados de problema Max-Cut con configuración completa de 5 nodos con 25 divisiones.

para esta subsección.

Los primeros resultados generados para esta instancia corresponden a 25 divisiones usando el modelo simple de QAOA con solo un parámetro de operador de fase y un parámetro de operador mezclador.

Los resultados de la **Tabla 4.21** muestran que el valor VEE de 3.514 se encuentra bastante lejos del valor óptimo esperado (usando función de costo) de 6, sin embargo, el estado obtenido (el estado que se presentó con mayor probabilidad) es uno de los estados óptimos del sistema.

Viendo la representación de la mejor combinación de parámetros en la **Figura 4.27** se tienen dos estados óptimos con probabilidad de 31 % para $|01010\rangle$ y 28 % para $|01011\rangle$ que en conjunto tienen una probabilidad cercana al 50 %. Este problema en específico tiene múltiples respuestas "correctas", estados que son óptimos debido a que presentan el mayor número de conexiones que es 6, esto da bastantes respuestas que pueden ser buenas dado que no es necesario obtener un estado en concreto porque se pueden obtener múltiples estados mientras que estos sean los estados óptimos.

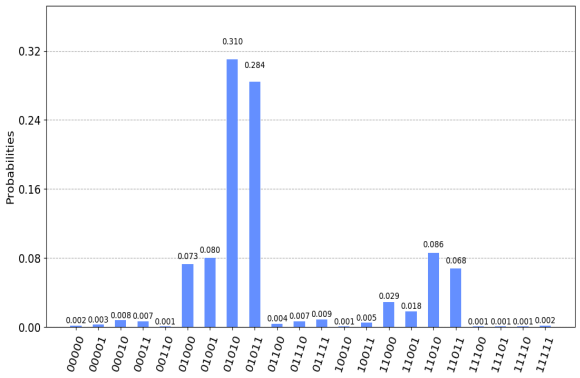


Figura 4.27: Histograma de mejor combinación de parámetros $\gamma = 1.256637\pi$ y $\beta = 4.775220\pi$ del problema de 5 nodos en configuración completa.

Estado obtenido	$\|01101\rangle$
VEE	3.634
Valor de γ	5.529203 π
Valor de β_1	4.775220 π
Valor de β_2	0.0 π

Tabla 4.22: Resultados de problema Max-Cut con configuración completa de 5 nodos con 25 divisiones (dos parámetros β).

Aumento de complejidad de operador mezclador usando dos parámetros β_1 y β_2

Con lo mencionado anteriormente, se propone aumentar la complejidad del operador mezclador y registrar si existe o no una mejora a la respuesta del sistema generada previamente.

Los resultados obtenidos son usando 25 divisiones para el rango de valores de los tres parámetros.

En la **Tabla 4.22** tenemos un pequeño aumento del valor de VEE, el estado con mayor número de apariciones es $|01101\rangle$ que es distinto al estado anterior, pero sigue siendo uno de los estados óptimos del problema.

Con la información de la **Figura 4.28** podemos ver que no existe un estado predominante en el sentido de que se presente con una probabilidad de medición considerablemente superior a los demás estado, sin embargo, todos los estados que se presentan por encima del 5 % son estados óptimos, si sumamos todas las probabilidades de estos estamos tenemos poco más del 98 % con respecto a los 1000 experimentos realizados, esto nos garantiza que a pesar de no tener un estado predominante cualquiera de los estados obtenidos (mayores al 5 %) son

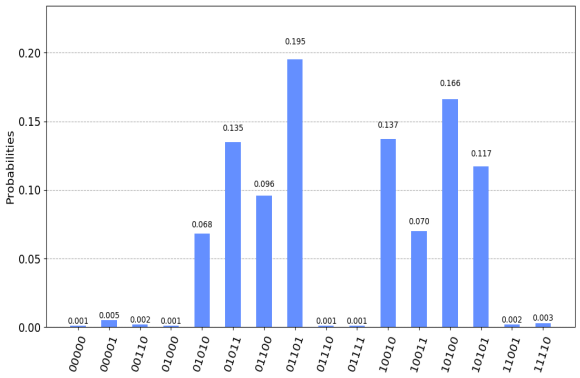


Figura 4.28: Histograma de mejor combinación de parámetros $\gamma = 5.529203\pi$, $\beta_1 = 4.775220\pi$ y $\beta_2 = 0.0\pi$ del problema de 5 nodos en configuración completa.

Estado obtenido	$\|01101\rangle$
VEE	3.65
Valor de γ_1	0.753982 π
Valor de γ_2	6.031857 π
Valor de β_1	1.507964 π
Valor de β_2	0.0 π

Tabla 4.23: Resultados de problema Max-Cut con configuración completa de 5 nodos con 25 divisiones (dos parámetros γ y dos parámetros β).

estados óptimos.

Aumento a dos parejas de parámetros γ_1 y γ_2, con β_1 y β_2

Ahora se usará modelo más complejo de QAOA en el problema de 5 nodos en configuración completa usando dos operadores de fase y dos operadores mezcladores (cuatro parámetros de QAOA).

De igual forma que las simulaciones anteriores se usarán 25 divisiones para el rango de valores de los cuatro parámetros.

Los resultados de la **Tabla 4.23** nos muestran una pequeña mejora del VEE, pero el factor más interesante es la prevalencia del mismo estado $|01101\rangle$ como estado óptimo, ya que al existir más estados óptimos es curioso que el mismo estado aparezca con mayor probabilidad.

Al igual que en la gráfica anterior, la **Figura 4.29** nos muestra que todos los estados con alta probabilidad (por encima del 5 %) son estados óptimos del sistema, con una probabilidad total del 99 %. Realmente no existe una diferencia sustancial entre la solución anterior con solo dos parámetros β y esta solución,

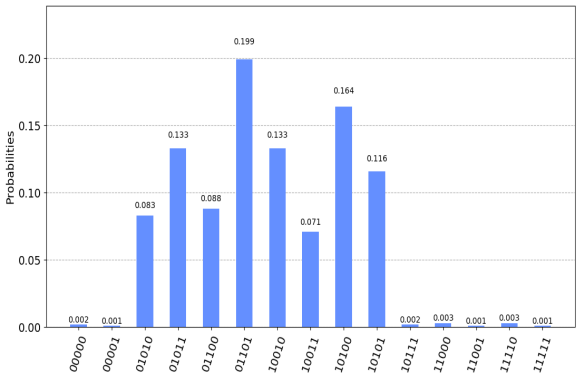


Figura 4.29: Histograma de mejor combinación de parámetros $\gamma_1 = 0.753982\pi$, $\gamma_2 = 6.031857\pi$, $\beta_1 = 1.507964\pi$ y $\beta_2 = 0.0\pi$ del problema de 5 nodos en configuración completa.

por lo que para esta instancia podemos argumentar que con el solo aumento de complejidad del operador mezclador se tiene una respuesta óptima y no es necesario agregar un parámetro extra de γ.

4.1.2. Algoritmo QAOA usando Búsqueda Local Iterativa

La heurística de **Búsqueda Local Iterativa** o Iterada (BLI) representa una alternativa al método de optimización de parámetros en QAOA, este método consiste en establecer "saltos" de diferentes magnitudes, empezando con un punto aleatorio dentro del espacio de búsqueda, pasando a un salto "mediano" (en este salto se toma en cuenta el mejor punto guardado más una perturbación, es aquí donde se asocia el término de *local*) y por último se realizan pequeños saltos aleatorios que corresponden a la implementación del *Ascenso en la Montaña Estocástico* (AME) que genera una respuesta con respecto a los valores explorados de esa pequeña zona; este proceso se repite con un número de reinicios predeterminado (saltos medianos) done se va evalúa cada respuesta generada de AME (saltos pequeños) y se almacena la mejor respuesta en cada etapa con respecto a la mejor respuesta conocida (mejor respuesta previa a esa iteración) hasta terminar todos los reinicios.

Además de las estrategias básicas de BLI, se incorporó una estrategia más para buscar hacer más efectiva la heurística. La estrategia consiste en evaluar si existe una convergencia temprana de la zona de búsqueda, esto se hace preguntando cada cierto número de evaluaciones de la BLI si existe una varianza de los datos, en caso de que esta varianza este por debajo de un valor gatillo se realiza un reinicio aleatorio de la búsqueda (si la varianza es mayor al valor gatillo no se realiza reinicio alguno), este reinicio se da evaluando una combinación aleatoria de

Estado obtenido	$\|001\rangle$
VEE	-2.9734
Valor de γ	1.403145 π
Valor de β	1.480049 π

Tabla 4.24: Resultados de problema ISM con configuración lineal de 3 partículas (para BLI).

parámetros para los operadores de fase y mezclador, si este nuevo valor aleatorio es mejor al anterior entonces se vuelve el nuevo mejor punto para reiniciar la exploración junto con su valor de evaluación y valores de mediciones, y en caso contrario se mantiene la mejor evaluación global (anterior) y solo se considera el mejor punto aleatorio encontrado para reiniciar la exploración; esto nos asegura mantener el mejor valor global mientras se evita que la búsqueda converja de manera temprana.

Simulaciones usando BLI para ISM

El método de BLI será usando para optimizar los parámetros del algoritmo QAOA en las instancias de ISM, estos resultados podrán ser comparados con los generados mediante BE y entender las ventajas y desventajas de ambos métodos.

Simulación usando BLI para configuración lineal de 3 partículas

El primer problema simulado usando BLI es el de 3 partículas en configuración lineal, los parámetros usados para BLI son: 100 reinicios, con un rango de perturbación de $[0, 1.0]$ con distribución uniforme, con un tamaño de paso para AME de $[-0.05, 0.05]$ con distribución uniforme y 1000 iteraciones. Para la estrategia adicional dentro de BLI se tienen un cálculo de varianza para 10 reinicios, con un valor gatillo de 1.0 y con 1000 posibles saltos aleatorios de búsqueda como nuevo inicio posible.

Los primeros resultados de BLI se aprecian en la **Tabla 4.24**, donde el valor de energía esperado fue de -2.9734 que no está muy cerca del valor esperado óptimo, sin embargo, el estado con mayor número de apariciones fue el estado $|001\rangle$ que corresponde al estado óptimo de este problema.

Viendo los datos estadísticos generados de la mejor combinación de parámetros en la **Figura 4.30** vemos que existe una probabilidad por encima del 80 % de medición del estado óptimo con respecto de los 1000 experimentos generados, estos resultados comparados con los obtenidos en para las **Figuras 4.4** y **4.5** (mejores resultados para BE de ISM 3 partículas) muestran una pequeña mejora usando BLI lo cual es un buen indicador para siguientes simulaciones.

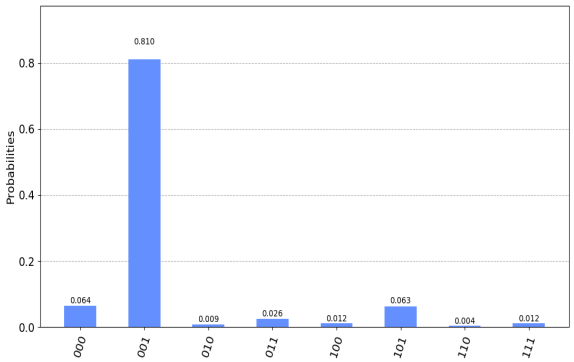


Figura 4.30: Histograma de mejor combinación de parámetros $\gamma = 1.403145\pi$ y $\beta = 1.480049\pi$ del problema de 3 partículas en configuración lineal (para BLI).

Estado obtenido	$\|0101\rangle$
VEE	-4.8908
Valor de γ	4.933719 π
Valor de β	1.615047 π

Tabla 4.25: Resultados de problema ISM con configuración cíclica de 4 partículas (para BLI).

Simulación usando BLI para configuración cíclica de 4 partículas

Para esta instancia del problema ISM para 4 partículas usando BLI se usaron los mismos valores de referencia de BLI que en la simulación anterior.

Los resultados de la **Tabla 4.25** se muestran con diferencias notables con respecto a los casos contemplando solo un parámetro γ y un parámetro β de las simulaciones usando BE (véase en las **Figuras 4.7** y **4.8**). En este caso vemos que existe un estado predominante $|0101\rangle$, este estado no es el óptimo, pero tiene un valor cercano de -4.9 (que correspondería al tercer mejor estado en relación con su costo).

Complementando la tabla anterior se tiene la gráfica de la **Figura 4.31** que nos muestra las probabilidades de medición de los estados para la mejor combinación de parámetros, se tiene una probabilidad muy baja de aparición del estado óptimo $|1101\rangle$ con apenas el 1.3 %, esto nos indica que se necesita modificar el modelo de QAOA para aumentar esta probabilidad de medición para el estado óptimo.

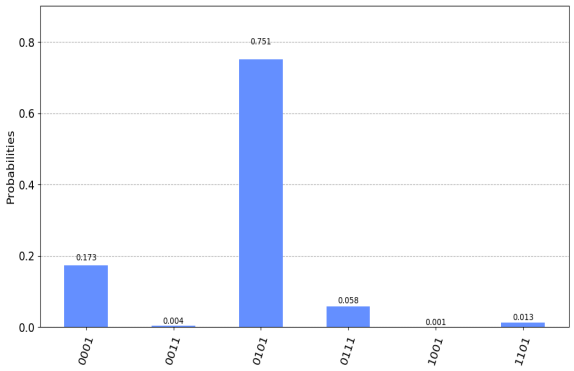


Figura 4.31: Histograma de mejor combinación de parámetros $\gamma = 4.933719\pi$ y $\beta = 1.615047\pi$ del problema de 4 partículas en configuración cíclica (para BLI).

Estado obtenido	$\|1101\rangle$
VEE	-4.0956
Valor de γ	3.809228 π
Valor de β_1	4.577415 π
Valor de β_2	3.074400 π

Tabla 4.26: Resultados de problema ISM con configuración cíclica de 4 partículas (para BLI) con β_1 y β_2.

Aumento de complejidad de operador mezclador usando dos parámetros β_1 y β_2

Al igual que en las estrategias anteriores de modificación de QAOA el primer paso (después de buscar aumentar la precisión de los valores de los parámetros) es agregar un parámetro extra dentro del operador mezclador.

La siguiente tabla muestra los resultados generados para la primera simulación modificando el algoritmo QAOA, los parámetros correspondientes a BLI son los mismos que para las simulaciones anteriores, esto se hace para mantener una relación comparable entre las diferentes simulaciones.

Ahora al ver los datos de la **Tabla 4.26** vemos que el valor de VEE se acerca bastante a un resultado adecuado, ya que el estado con mayor probabilidad de aparición es el estado óptimo $|1101\rangle$ con una probabilidad superior al 60 %, también es bastante bueno que el siguiente estado con mayor probabilidad de medición es el estado $|0001\rangle$ que es el estado más cercano al óptimo del sistema.

Otro aspecto importante a señalar es que el aumento de un nuevo parámetro no hizo que aumentara el costo computacional del algoritmo de forma considerable porque BLI genera una elección aleatoria de la variable que será alterada

(γ, β_1 o β_2) y esta variable se modifica con una cierta distribución para generar tanto la Búsqueda Local como la etapa de Ascenso en la Montaña Estocástico, entonces debido a que la calibración de BLI se mantiene entre simulaciones, el hecho de agregar una nueva variable para explorar solo representa un valor de elección posible para el algoritmo a diferencia de Búsqueda Exhaustiva donde representaba un aumento de las posibles combinaciones de resultados.

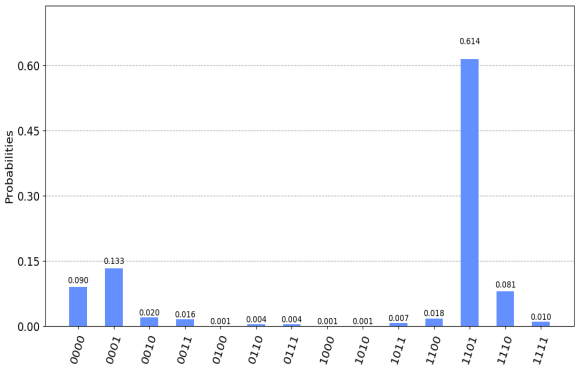


Figura 4.32: Histograma de mejor combinación de parámetros $\gamma = 4.985547\pi$, $\beta_1 = 4.637650\pi$ y $\beta_2 = 0.012478\pi$ del problema de 4 partículas en configuración cíclica (para BLI).

La gráfica de la **Figura 4.32** nos muestra como el aumento del parámetro β_2 aumento la probabilidad de medición del estado $|1101\rangle$, pero este resultado aún puede ser mejorado modificando el modelo de QAOA.

Estos resultados comparados con los obtenidos en el método de BE son significativamente peores en lo que se refiere al estado con mayor probabilidad de medición, pero también son más eficientes en el tiempo y recursos necesarios para alcanzarlos, y dado que el aumento de una variable no representa un aumento exorbitante en recursos para simular en el método BLI ahora podemos analizar el caso del aumento a dos operadores de fase y dos operadores mezcladores dentro de QAOA.

Aumento a dos parejas de parámetros γ_1 y γ_2, con β_1 y β_2

El aumento para dos parejas de operadores de fase y mezclador se realiza como un método para garantizar que QAOA sea capaz explorar la mayor cantidad de combinaciones de estados dentro del espacio de búsqueda. La simulación para esta modificación de QAOA utilizan los mismos parámetros para BLI que en los experimentos anteriores.

En este caso, los resultados de la **Tabla 4.27** nos muestran que de nuevo se obtuvo como estado con mayor probabilidad a $|1101\rangle$ que es el estado óptimo,

Estado obtenido	$\|1101\rangle$
VEE	-4.7374
Valor de γ_1	3.737709 π
Valor de γ_2	5.034290 π
Valor de β_1	4.652072 π
Valor de β_2	3.813496 π

Tabla 4.27: Resultados de problema ISM con configuración cíclica de 4 partículas (para BLI) con γ_1 y γ_1, β_1 y β_2.

además se tiene un de VEE de -4.7374 se encuentra más cerca del valor VEE evaluado con el estado óptimo (teniendo una probabilidad del 100 %), esto ya muestra una mejora con respecto del método anterior.

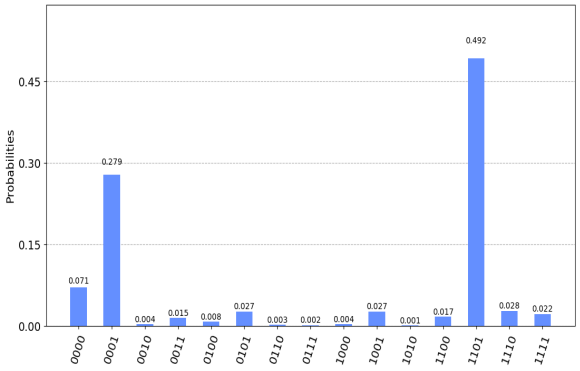


Figura 4.33: Histograma de mejor combinación de parámetros $\gamma_1 = 3.737709\pi$, $\gamma_2 = 5.034290\pi$, $\beta_1 = 4.652072\pi$ y $\beta_2 = 3.813496\pi$ del problema de 4 partículas en configuración cíclica (para BLI).

En la gráfica de la **Figura 4.33** vemos que existe un decremento de la aparición del estado óptimo con un cercano 50 % si se compara con el método anterior, también se ve un aumento en la probabilidad de aparición del segundo mejor estado $|0001\rangle$, si tomamos la probabilidad total de apariciones de ambos estados es de más del 75 %. Algo importante resaltar es que el VEE mejoró y esto nos indica que la distribución de probabilidades de medición de los estados es mejor, esto se interpreta con que existe menos probabilidad de medición de estados no deseados y se tiene una prevalencia del estado óptimo y los cercanos a este, esto es lo que se busca debido a que QAOA pretende dar una solución donde se obtenga el resultado óptimo con alta probabilidad, pero debido a que es un algoritmo de aproximación no se descarta la existencia de mediciones hacia otros estados, y entre más cercano sea el valor de VEE al del estado óptimo esto garantizará que las observaciones se mantengan en los mejores estados del sistema.

Estado obtenido	$\|00000\rangle$
VEE	-8.3734
Valor de γ	5.048866 π
Valor de β	4.685957 π

Tabla 4.28: Resultados de problema ISM con configuración completa de 5 partículas (para BLI).

Simulación usando BLI para configuración completa de 5 partículas

Para esta instancia se usaron los mismos valores de configuración para BLI que en casos anteriores. El primer experimento realizado fue usando la configuración básica de QAOA con un parámetro γ y uno β.

Viendo los resultados de la **Tabla 4.28** existe una gran similitud con los generados por BE en las **Tablas 4.11** y **4.12**, pero en este caso tenemos un mejor valor de VEE con -8.3734 que se encuentra más cercano al valor de costo evaluado para el estado óptimo.

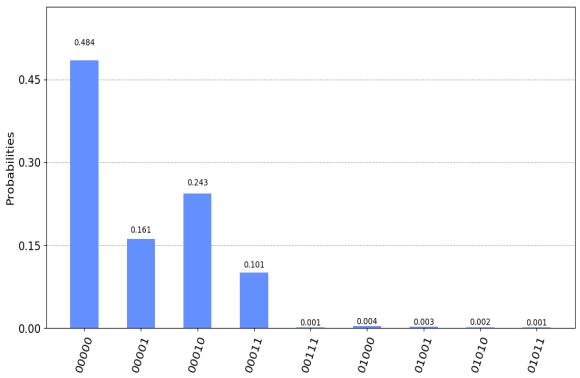


Figura 4.34: Histograma de mejor combinación de parámetros $\gamma = 5.048866\pi$ y $\beta = 4.685957\pi$ del problema de 5 partículas en configuración completa (para BLI).

En adición a los datos de la tabla, la **Figura 4.34** coincide con los resultados observados en las simulaciones anteriores de BE.

Aumento de complejidad de operador mezclador usando dos parámetros β_1 y β_2

Ahora como siguiente paso para mejorar la respuesta de QAOA es necesario modificar el modelo del algoritmo, el primer experimento se realiza aumentando la

Estado obtenido	$\|00000\rangle$
VEE	-7.3026
Valor de γ	5.031833 π
Valor de β_1	4.710976 π
Valor de β_2	6.245924 π

Tabla 4.29: Resultados de problema ISM con configuración completa de 5 partículas con dos βs (para BLI).

complejidad del operador mezclador, y se usarán los mismos valores de calibración para BLI.

Los resultados de esta simulación se muestran en la **Tabla 4.29**, en primera instancia vemos que el estado con mayor probabilidad es el estado óptimo $|00000\rangle$ pero con un valor de VEE más alejado del valor para el estado óptimo, lo cual es bastante interesante porque se esperaría que el resultado de VEE estuviera más cercano al óptimo.

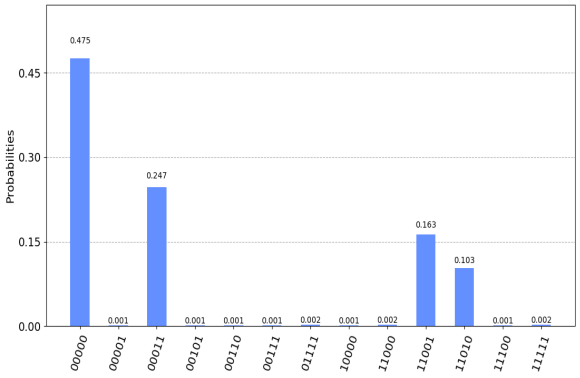


Figura 4.35: Histograma de mejor combinación de parámetros $\gamma = 5.031833\pi$, $\beta_1 = 4.710976\pi$ y $\beta_2 = 6.245924\pi$ del problema de 5 partículas en configuración completa con dos βs (para BLI).

Agregando el histograma de la **Figura 4.35** vemos que existe un deterioro en la calidad de la respuesta con relación a la simulación anterior con solo un par de parámetros, ya que si bien la probabilidad de medición del estado óptimo es casi la misma que en la simulación anterior, la probabilidad de medición de los estados no deseados es peor, ya que los estados presentan una probabilidad de medición por encima del 10 % y son estados muy alejados del óptimo.

Estado obtenido	$\|00000\rangle$
VEE	-7.3473
Valor de γ_1	5.744897 π
Valor de γ_2	0.217068 π
Valor de β_1	1.552818 π
Valor de β_2	3.193465 π

Tabla 4.30: Resultados de problema ISM con configuración completa de 5 partículas con dos γs y dos βs (para BLI).

Aumento a dos parejas de parámetros γ_1 y γ_2, con β_1 y β_2

Como siguiente prueba se tiene el aumento a dos parejas de parámetros para el modelo de QAOA, esta estrategia sigue los mismos valores usados para BLI.

La **Tabla 4.30** nos muestra una pequeña mejora en el valor VEE que se encuentra más cercano al valor óptimo comparado con el caso anterior, de igual forma el estado con mayor probabilidad de se mantiene como el estado óptimo.

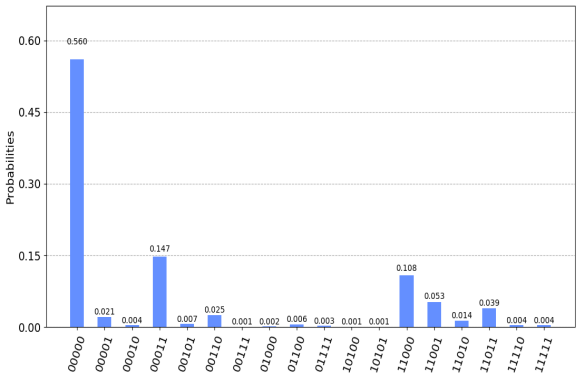


Figura 4.36: Histograma de mejor combinación de parámetros $\gamma_1 = 5.744897\pi$, $\gamma_2 = 0.217068\pi$, $\beta_1 = 1.552818\pi$ y $\beta_2 = 3.193465\pi$ del problema de 5 partículas en configuración completa con dos γs y dos βs (para BLI).

Anexando al análisis los datos de la **Figura 4.36** podemos observar que existe una mejora en la probabilidad de medición del estado óptimo con casi 60 %, sin embargo, vemos que aún no existe una respuesta que pueda considerarse aceptable, esto es bastante extraño dadas las simulaciones anteriores, aunque esto se puede relacionar con el aumento de dimensiones de los posibles parámetros que genera que la BLI sea menos eficiente debido al número de iteraciones que se permiten tanto de reinicios como de pasos dentro del AME. Esto podría ser uno de los resultados más interesantes, ya que cada parámetro correspondiente a un operador de fase u operador mezclador es (en teoría) continuo en el espectro

Estado obtenido	$\|010\rangle$
VEE	1.716
Valor de γ	0.573760 π
Valor de β	4.298174 π

Tabla 4.31: Resultados de problema Max-Cut con configuración lineal de 3 nodos (para BLI).

de 0 a 2π, entonces al ir aumentando más dimensiones al espacio de búsqueda es necesario contemplar el aumento de iteraciones para que BLI sea más eficiente y pueda cubrir una mayor área de búsqueda.

Simulaciones usando BLI para Max-Cut

Para el siguiente bloque de simulaciones se tienen las de Max-Cut, este modelo de problemas es bastante similar a los de ISM, sin embargo, dado que es un problema bastante estudiado es interesante explorarlo y analizarlo con una combinación de BLI y la implementación de QAOA.

Simulación usando BLI para configuración lineal de 3 nodos

El primer problema abordado será la configuración lineal de 3 nodos para Max-Cut, este problema usará los mismos valores para BLI que en las simulaciones de ISM, estos valores son: 100 reinicios, con un rango de perturbación de $[0, 1.0]$ con distribución uniforme, tamaño de paso para AME de $[-0.05, 0.05]$ con distribución uniforme y 1000 iteraciones. Para la estrategia adicional dentro de BLI se tienen un cálculo de varianza para 10 reinicios, con un valor gatillo de 1.0 y con 1000 posibles saltos aleatorios de búsqueda como nuevo inicio posible.

Los resultados de la **Tabla 4.31** nos muestran que existe una pequeña mejoría con respecto al resultado obtenido para el método de BE en la **Tabla 4.17**, también se tiene un cambio en lo que respecta al valor del parámetro β que es de esperarse debido al aumento del valor VEE.

Ahora con el histograma de la **Figura 4.37** vemos que efectivamente existe una pequeña mejoría con respecto al planteamiento del problema con BE, esto resulta en una probabilidad total de medición para los estados óptimos $|010\rangle$ y $|101\rangle$ de 74.5 % del total de experimentos.

Simulación usando BLI para configuración cíclica de 4 nodos

La siguiente instancia simulada es el problema de 4 nodos en configuración cíclica, esta simulación usa los mismos parámetros de calibración para BLI y QAOA que se usaron en los casos anteriores.

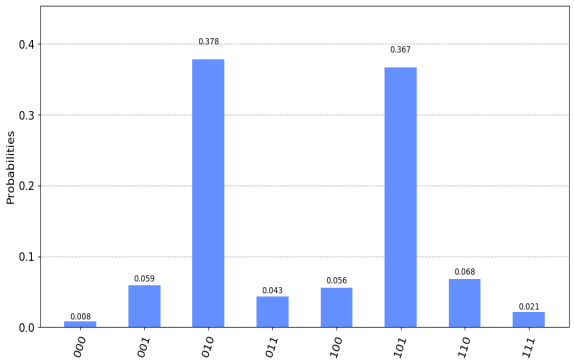


Figura 4.37: Histograma de mejor combinación de parámetros $\gamma = 0.573760\pi$ y $\beta = 4.298174\pi$ del problema de 3 nodos en configuración lineal (para BLI).

Estado obtenido	$\|1010\rangle$
VEE	2.144
Valor de γ	6.255068 π
Valor de β	0.098935 π

Tabla 4.32: Resultados de problema Max-Cut con configuración cíclica de 4 nodos (para BLI).

Analizando los datos presentes en la **Tabla 4.32** vemos que el valor de VEE es bastante similar al obtenido para el método de Búsqueda Exhaustiva (que puede verse en la **Tabla 4.18**), VEE tiene un incremento leve usando BLI, pero aun el valor está muy lejos de la evaluación para el estado óptimo.

Vemos que el patrón de distribución aleatorio relativamente uniforme se repite en la **Figura 4.38** que se presentó también en BE para el mismo problema, esto nos indica que a pesar del cambio de método de optimización no es posible alcanzar un estado óptimo del sistema con este modelo de QAOA con dos parámetros, uno para el operador de fase y uno para el operador mezclador.

Aumento de complejidad de operador mezclador usando dos parámetros β_1 y β_2

La primera modificación probada en QAOA fue aumentar la complejidad del operador mezclador, al igual que en las simulaciones anteriores los valores de calibración para QAOA y BLI son los mismos.

Los resultados de la **Tabla 4.33** muestran un aumento considerable (más cercanos al óptimo) con respecto al modelo básico de QAOA, pero estos resultados también son muy similares a los del método por BE.

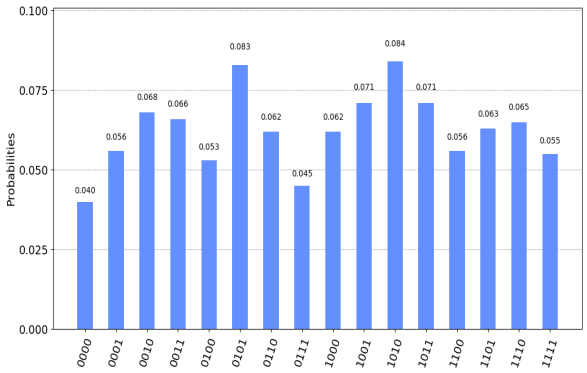


Figura 4.38: Histograma de mejor combinación de parámetros $\gamma = 6.255068\pi$ y $\beta = 0.098935\pi$ del problema de 4 nodos en configuración cíclica (para BLI).

Estado obtenido	$\|0101\rangle$
VEE	2.6919
Valor de γ	2.698002 π
Valor de β_1	5.458207 π
Valor de β_2	5.488248 π

Tabla 4.33: Resultados de problema Max-Cut con configuración cíclica de 4 nodos con dos parámetros β (para BLI).

Para los datos de la **Figura 4.39** vemos que empieza a presentarse un aumento en la probabilidad de medición de los estados óptimos $|0101\rangle$ y $|1010\rangle$ con respecto de los otros estados, y de nueva forma vemos que los resultados de BE y estos usando BLI son prácticamente idénticos, lo que nos puede sugerir que usar BLI para este problema es más eficiente que BE tomando en cuenta los recursos computacionales y la complejidad de ambos métodos.

Aumento a dos parejas de parámetros γ_1 y γ_2, con β_1 y β_2

El último modelo simulado para el problema de 4 nodos se realizó con dos parejas de parámetros, este modelo para el caso de BE era el más costoso en términos computacionales, pero como se comentó anteriormente, para BLI realmente no representa un aumento considerable de complejidad debido a que los valores de calibración para BLI se mantienen.

La **Tabla 4.34** tiene como resultado para VEE el valor exacto de la evaluación para cualquiera de los dos estados óptimos con 4, esto es algo bastante importante y significativo, ya que es el primer resultado (simulándolo de manera local) que tiene como Valor de Energía Esperado el costo exacto óptimo esperado.

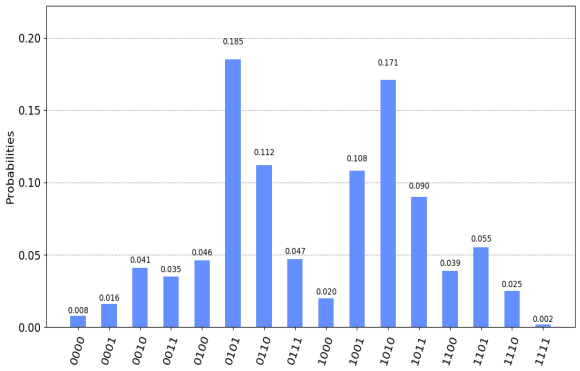


Figura 4.39: Histograma de mejor combinación de parámetros $\gamma = 2.698002\pi$, $\beta_1 = 5.458207\pi$ y $\beta_2 = 5.488248\pi$ del problema de 4 nodos en configuración cíclica (para BLI).

Estado obtenido	$\|0101\rangle$
VEE	4
Valor de γ_1	1.564337 π
Valor de γ_2	3.939548 π
Valor de β_1	5.315979 π
Valor de β_2	2.370957 π

Tabla 4.34: Resultados de problema Max-Cut con configuración cíclica de 4 nodos con dos parámetros β y dos parámetros γ (para BLI).

La gráfica de la **Figura 4.40** muestra que el estado óptimo $|0101\rangle$ se presenta con el 100 % de probabilidad, este es el primer resultado que nos garantiza que la combinación de parámetros $\gamma_1 = 1.564337\pi$, $\gamma_2 = 3.939548\pi$, $\beta_1 = 5.315979\pi$ y $\beta_2 = 2.370957\pi$ genera una respuesta óptima exacta. Este resultado debe ser analizado para su simulación en un ordenador cuántico, ya que en el estado actual del hardware cuántico existe una gran cantidad de elementos que pueden generar ruido y generar estados erróneos en el proceso de medición. Otro resultado relevante es que al generar esta simulación existieron muchas combinaciones de resultados que generaban el estado óptimo $|0101\rangle$ o $|1010\rangle$, esto nos dice que existe una "colección" de valores que mapean desde el espacio de búsqueda de los parámetros al espacio de Hilbert donde existen los estados del sistema.

Simulación usando BLI para configuración completa de 5 nodos

El último problema a simular localmente usando BLI en Max-Cut es el caso de 5 nodos en configuración completa, y en donde se usaron los mismos valores de calibración para BLI y QAOA.

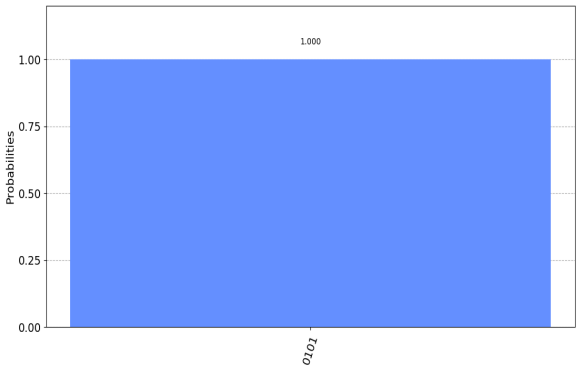


Figura 4.40: Histograma de mejor combinación de parámetros $\gamma_1 = 1.564337\pi$, $\gamma_2 = 3.939548\pi$, $\beta_1 = 5.315979\pi$ y $\beta_2 = 2.370957\pi$ del problema de 4 nodos en configuración cíclica (para BLI).

Estado obtenido	$\|10101\rangle$
VEE	3.6879
Valor de γ	4.968460 π
Valor de β	4.724006 π

Tabla 4.35: Resultados de problema Max-Cut con configuración completa de 5 nodos (para BLI).

La **Tabla 4.35** nos muestra datos relativamente similares a los resultantes usando el método de BE, aunque el valor de VEE está un poco más cercano al óptimo (valor de evaluación para alguno de los estados óptimos es de 6) y existe una diferencia notable respecto al valor de γ con respecto al establecido para BE.

La **Figura 4.41** nos muestra resultados diferentes y a la vez similares al caso de BE, son diferentes porque el estado con mayor probabilidad es diferente al de caso de BE, pero también se sabe que este problema, debido a la configuración completa, posee muchos estados óptimos los cuales se presentan para este caso en los estados $|10100\rangle$, $|10101\rangle$ y $|10110\rangle$ (que es un comportamiento parecido al de los resultados de BE), siendo solo el estado $|10111\rangle$ uno de los estados no deseados que se tienen con una probabilidad de medición por encima del 8 %.

Aumento de complejidad de operador mezclador usando dos parámetros β_1 y β_2

El siguiente paso que puede probarse para este problema es modificar el modelo de QAOA, ya que esto puede generar que se tenga un mejor respuesta, donde se pretende aumentar la probabilidad de aparición para los estados óptimos y

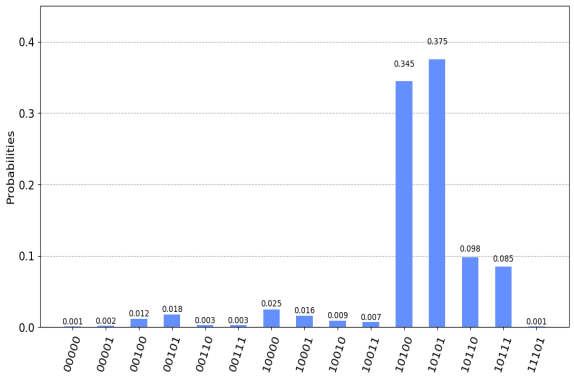


Figura 4.41: Histograma de mejor combinación de parámetros $\gamma = 4.968460\pi$ y $\beta = 4.724006\pi$ del problema de 5 nodos en configuración completa (para BLI).

Estado obtenido	$\|01011\rangle$
VEE	3.7499
Valor de γ	5.586943 π
Valor de β_1	4.693765 π
Valor de β_2	3.122219 π

Tabla 4.36: Resultados de problema Max-Cut con configuración completa de 5 nodos para dos parámetros β (para BLI).

decrementar la probabilidad para los estados no deseados.

Revisando la **Tabla 4.36** vemos que el aumento a dos parámetros para el operador mezclador genera un pequeño incremento para VEE, comparando este resultado con el caso de BE de la **Tabla 4.22** se tienen valores similares en cuanto VEE, sin embargo, los demás elementos como el estado óptimo y los parámetros de los operadores difieren bastante.

El histograma de la **Figura 4.42** nos confirma que al igual que para BE si sumamos los estados con una probabilidad de 2 % o más tenemos que los estados óptimos (estados que evalúan a 6 como el número máximo de conexiones entre los dos subconjuntos) nos dan una probabilidad total de medición de aproximadamente el 95 %.

Aumento a dos parejas de parámetros γ_1 y γ_2, con β_1 y β_2

Finalmente, como última simulación para el problema de 5 nodos (y de la subsección de simulaciones locales) se altera de nueva forma a QAOA agregando dos parámetros (dos operadores fase y dos mezcladores) uno γ y uno β.

La información mostrada en la **Tabla 4.37** nos expone un valor de VEE

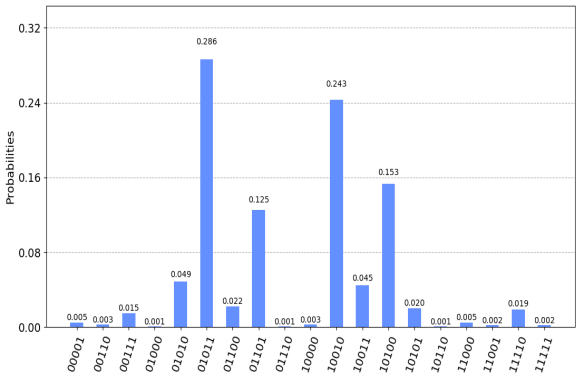


Figura 4.42: Histograma de mejor combinación de parámetros $\gamma = 5.586943\pi$, $\beta_1 = 4.693765\pi$ y $\beta_2 = 3.122219\pi$ del problema de 5 nodos en configuración completa (para BLI).

Estado obtenido	$\|01011\rangle$
VEE	3.758
Valor de γ_1	3.849325 π
Valor de γ_2	2.737281 π
Valor de β_1	1.601462 π
Valor de β_2	3.103111 π

Tabla 4.37: Resultados de problema Max-Cut con configuración completa de 5 nodos para dos parámetros γ y dos parámetros β (para BLI).

prácticamente idéntico en relación con la modificación anterior de QAOA (con dos parámetros β solamente), en comparación con el método de BE vemos que los resultados de ambas tablas tiene valores cercanos de VEE siendo el método por BLI el que presenta un mejor valor, el estado óptimo y los parámetros de los operadores tienen valores completamente distintos entre sí.

Y finalmente, en la **Figura 4.43** se observa una distribución de la probabilidad de medición de los estados óptimos bastante parecida a la del modelo anterior de dos βs, las diferencias que se presentan son con relación al aumento o disminución de aparición sobre un estado contra otro, pero en general se tiene un histograma bastante similar con una pequeña ventaja para BLI. Analizando este resultado contra el método de BE vemos que la probabilidad total de medición de ambos respecto a los estados óptimos es bastante cercana, pero, por otro lado, el método de BE presenta una mejor distribución porque existen estados no deseados con probabilidades menores al 1 % e incluso estados no deseados que no presentan probabilidad de medición alguna.

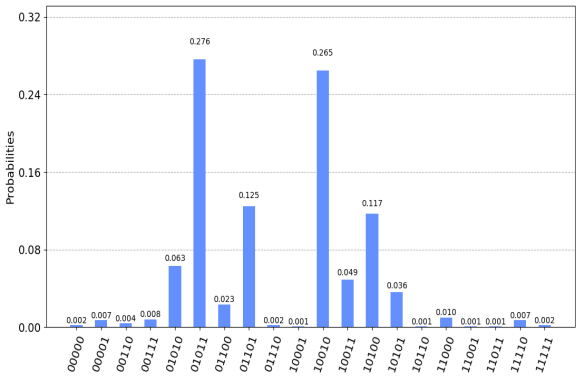


Figura 4.43: Histograma de mejor combinación de parámetros $\gamma_1 = 3.849325\pi$, $\gamma_2 = 2.737281\pi$, $\beta_1 = 1.601462\pi$ y $\beta_2 = 3.103111\pi$ del problema de 5 nodos en configuración completa (para BLI).

4.2. Simulaciones usando IBMQ

El siguiente factor experimentado se refiere al uso de una computadora cuántica real y los efectos que esta tiene en comparación con los resultados simulados en una computadora clásica que solo puede aproximar el comportamiento cuántico. Esto es de relevancia para cualquier estudio de cómputo cuántico experimental, ya que nos muestra en que estado se encuentra el hardware cuántico actual, este hardware cuántico puede presentar problemas de ruido, decoherencia y otros fenómenos que pueden afectar al resultado, generando variaciones entre las simulaciones locales y las realizadas en el dispositivo físico.

Debido al tipo de cuenta que se posee actualmente en IBM Q solo es posible emplear computadoras cuánticas de hasta 5 qubits, estas computadoras pueden variar en el número de compuertas cuánticas que permiten implementar, este número puede ser 8, 16 o 32 que tiene una relación directa con el número de compuertas (que puede encontrarse como **volumen cuántico**) que pueden usarse, por lo anterior mencionado se optará por solo usar computadoras de 5 qubits con 32 de volumen cuántico, esto se hará acorde a su disponibilidad porque puede que algunas estén ocupadas al momento de buscar simular un programa. Las computadoras que cuentan con estas características son:

- *ibmq_manila*
  - Volumen cuántico: 32
  - Tipo de procesador: Falcon r5.11L
- *ibmq_bogota*

  - Volumen cuántico: 32
  - Tipo de procesador: Falcon r4L

- *ibmq_santiago*
  - Volumen cuántico: 32
  - Tipo de procesador: Falcon r4L

También, debido a la disponibilidad de tiempo y recursos que se permiten a cada usuario (ya que es un servicio libre y sin costo para fines académicos por parte de IBM) no es factible probar el desarrollo del proceso de optimización tanto para BE como para BLI, y simular cada combinación generada por estos métodos en QAOA, además, dado que es un proceso de optimización que se realiza de manera clásica no existe mucha diferencia si se realiza en otra computadora clásica. Entonces lo que se simulará para cada instancia de los problemas de ISM y Max-Cut en las simulaciones reales será usando la mejor combinación encontrada de los métodos BE y BLI en las simulaciones locales, esto nos mostrará si existen o no diferencias respecto a las simulaciones locales. Debe considerarse que no se podrá comprobar la ventaja más relevante del equipo cuántico que se muestra en el uso de varios qubits (problemas más completos), ya que para estos problemas a escala las simulaciones locales no tuvieron muchos problemas en completarse, por ello las diferencias que se esperan obtener serán directamente en los resultados y no en el desempeño del algoritmo.

4.2.1. Algoritmo QAOA usando Búsqueda Exhaustiva

De igual forma que en las simulaciones locales, se inicia simulando las mejores combinaciones de parámetros usando el método de Búsqueda Exhaustiva (BE) para los distintos problemas en las simulaciones reales.

Simulaciones usando BE para ISM

El método de BE en combinación con QAOA será usado para ISM donde se tratarán los problemas con los mismos valores y características que en el caso de las simulaciones locales. Los problemas que serán abordados son 3 partículas en configuración lineal, 4 partículas en configuración cíclica y 5 partículas en configuración completa.

Todas las simulaciones se harán usando solo la evaluación por VEE (Valor de Energía Esperado).

Estado obtenido	$\|001\rangle$
VEE	-2.64919
Valor de γ	1.507946 π
Valor de β	1.507946 π

Tabla 4.38: Resultados de problema ISM con configuración lineal de 3 partículas usando IBM Q.

Simulación usando BE para configuración lineal de 3 partículas

La primera instancia simulada será para 3 partículas en configuración lineal, usando BE (en las simulaciones locales) se obtuvieron los siguientes valores para los parámetros: 1.507946 para γ y 1.507946 para β, estos son los parámetros generados para la simulación de 25 divisiones (el rango de 25 divisiones se usará para todas las simulaciones reales de BE).

El valor VEE que se muestra en la **Tabla 4.38** es cercano a lo generado por la simulación local (obsérvese la **Tabla 4.17**) sin embargo, el valor de la simulación local es un poco más cercano al valor de evaluación esperado para el estado óptimo.

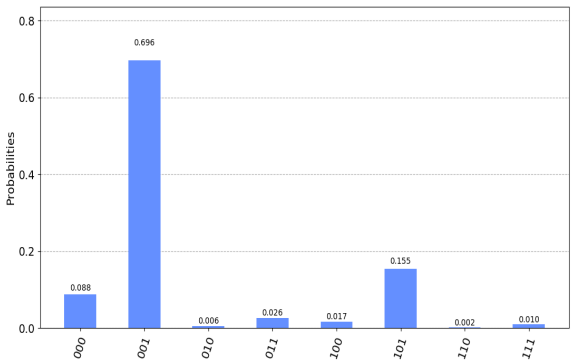


Figura 4.44: Histograma de mejor combinación de parámetros $\gamma = 1.507946\pi$ y $\beta = 1.507946\pi$ problema de 3 partículas en configuración lineal usando IBM Q.

Incluyendo los datos de la **Figura 4.44** vemos que la probabilidad de medición para ese estado óptimo $|001\rangle$ es cercana al 70 % pero con un decremento de la calidad de respuesta con respecto al 77 % generado en la simulación local, además se aprecia un incremento en la probabilidad de medición de los estados no deseados incluyendo al estado $|101\rangle$.

Estado obtenido	$\|0101\rangle$
VEE	-4.0834
Valor de γ	5.026548 π
Valor de β	1.759291 π

Tabla 4.39: Resultados de problema ISM con configuración cíclica de 4 partículas usando IBM Q.

Simulación usando BE para configuración cíclica de 4 partículas

De igual forma que en el caso anterior ahora tomamos la instancia para el problema de 4 partículas usando la mejor combinación de parámetros generada en la simulación local usando 25 divisiones.

La **Tabla 4.39** para el caso de 4 partículas nos muestra un deterioro en la respuesta para el valor de VEE comparado con la simulación local, en este caso se tiene un valor de -4.0834 que está muy lejano al valor de -4.8044 obtenido de manera local (dadas las escalas de valores que se están analizando).

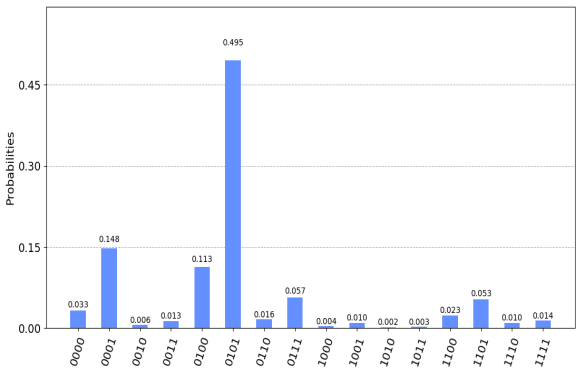


Figura 4.45: Histograma de mejor combinación de parámetros $\gamma = 5.026548\pi$ y $\beta = 1.759291\pi$ problema de 4 partículas en configuración cíclica usando IBM Q.

Incorporando el histograma presente en la **Figura 4.45** vemos que el estado con mayor probabilidad de aparición es $|0101\rangle$ (que no es el estado óptimo, pero se encuentra cerca del valor de evaluación para el estado óptimo) pero tiene una probabilidad muy baja con casi 50 % que es mucho menor a la obtenida de manera local, algo curioso es que existe un aumento de apariciones para el estado $|0001\rangle$ que es un mejor estado (respecto a su evaluación) que $|0101\rangle$.

Estado obtenido	$\|1101\rangle$
VEE	-1.7432
Valor de γ	3.769911 π
Valor de β_1	4.77522 π
Valor de β_2	3.518583 π

Tabla 4.40: Resultados de problema ISM con configuración cíclica de 4 partículas con dos parámetros β usando IBM Q.

Aumento de complejidad de operador mezclador usando dos parámetros β_1 y β_2

Modificando el modelo original de QAOA agregando un parámetro extra al operador mezclador (se adhiere usando entrelazamiento cuántico), los parámetros usados para γ, β_1 y β_2 son 3.769911, 4.775220 y 3.518583 respectivamente.

La **Tabla 4.40** muestra un valor de VEE peor al obtenido en la simulación local para la misma instancia con esa misma combinación de parámetros, sin embargo, aún se tiene el estado $|1101\rangle$ como el estado con mayor probabilidad de medición. Estos resultados fueron inicialmente simulados usando la computadora cuántica de IBM llamada **ibmq_lima**, pero se obtuvieron valores para VEE mucho más lejanos que el de la tabla anterior con respecto al valor de evaluación para el estado óptimo, después, al cambiar a la computadora **ibmq_santiago** se obtuvieron mejores resultados que se muestran en la tabla pasada; investigando un poco más sobre ambos dispositivos se tiene que para el estado actual (fecha: 6/10/21) de **ibmq_lima** se tiene un error de compuerta $CNOT$ de $1.931 \times e^{-2}$ y para la computadora **ibmq_santiago** un error de $7.585 \times e^{-3}$ (fecha: 6/10/21) lo cual se relaciona con la mejora en los resultados usando esta computadora de IBM Q, esto nos muestra diferencias entre los dispositivos donde se corre el programa cuántico, los cuales pueden inferir en la calidad del resultado según la calibración (el error respecto a $CNOT$, tiempo de ejecución sin decoherencia, etc.) y el modelo de la computadora cuántica (tipo de procesador cuántico principalmente).

Si comparamos los datos de la **Figura 4.46** con los obtenidos de la **Figura 4.13** existe un detrimento respecto a la probabilidad de medición del estado óptimo, ya que en este caso se presenta con solo 25 % y para la simulación local se presentó con casi 50 % que es el doble de probabilidad sobre la simulación en la computadora cuántica, además vemos que existe un aumento importante en los estados no deseados (excepto $|0001\rangle$ que es el estado más cercano al óptimo).

Simulación usando BE para configuración completa de 5 partículas

Pasando al problema de 5 partículas en configuración completa se usarán los valores de γ con 5.026548 y β con 4.775220, que son obtenidos del primer modelo

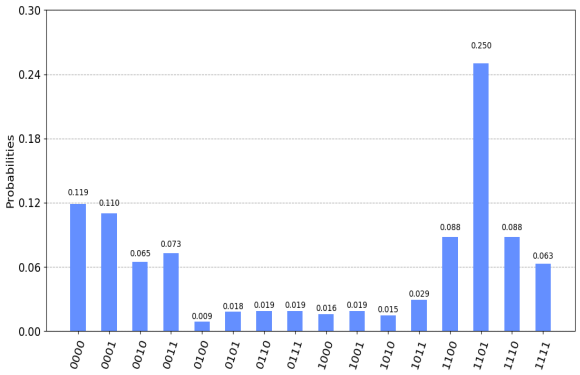


Figura 4.46: Histograma de mejor combinación de parámetros $\gamma = 3.769911\pi$, $\beta_1 = 4.77522\pi$ y $\beta_2 = 3.518583\pi$ problema de 4 partículas en configuración cíclica usando IBM Q.

Estado obtenido	$\|00000\rangle$
VEE	-3.3192
Valor de γ	5.026548 π
Valor de β	4.775220 π

Tabla 4.41: Resultados de problema ISM con configuración completa de 5 partículas usando IBM Q.

de QAOA usando 25 divisiones.

Los resultados de la **Tabla 4.41** nos muestran un decremento bastante significativo en el valor de VEE con respecto a la simulación local (que tuvo un resultado de -7.9012), pero aún se mantiene el estado óptimo $|00000\rangle$ como el estado con mayor probabilidad de aparición.

Con la información del histograma de la **Figura 4.47** vemos que efectivamente existe una respuesta muy lejana a la deseada, y aun comparándola con la gráfica de la simulación local vemos que no existe una relación entre ambos resultados a pesar de que en ambas simulaciones presentan al estado óptimo con mayor probabilidad de medición. Aquí también podemos entender porque existe un deterioro en VEE y es porque existe un aumento en la aparición de estados no deseados, esto incluye al peor estado $|10100\rangle$ que tiene una probabilidad de casi el 3 % pero tiene un aporte de costo de evaluación de 11.3.

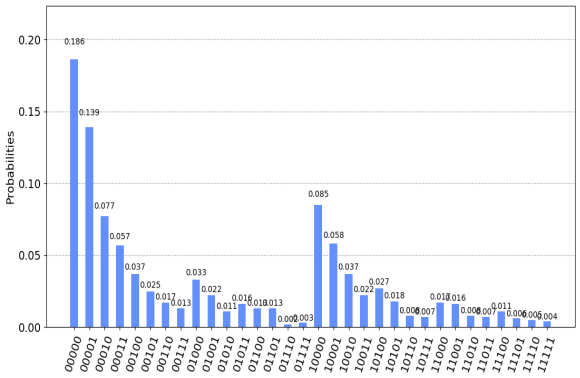


Figura 4.47: Histograma de mejor combinación de parámetros $\gamma = 5.026548\pi$ y $\beta = 4.775220\pi$ problema de 5 partículas en configuración completa usando IBM Q.

Estado obtenido	$\|00011\rangle$
VEE	-2.1216
Valor de γ	5.026548 π
Valor de β_1	4.523893 π
Valor de β_2	0.0 π

Tabla 4.42: Resultados de problema ISM con configuración completa de 5 partículas con dos parámetros β usando IBM Q.

Aumento de complejidad de operador mezclador usando dos parámetros β_1 y β_2

Con la modificación del operador mezclador usando dos parámetros β se usaron los siguientes valores para γ con 5.026548, para β_1 tenemos 4.523893 y para β_2 es 0.0

El valor obtenido de VEE de -2.1216 que se tiene en la **Tabla 4.42** nos presenta un valor más lejano (que en el caso anterior) del valor de evaluación para estado óptimo, esto muestra además una peor respuesta comparada con el método simple de QAOA usando solo dos parámetros y también es una respuesta muy alejada del valor generado para el mismo modelo con un parámetro γ y dos parámetros β en las simulaciones locales. También es importante mencionar que el estado con mayor probabilidad de aparición ya no es el estado óptimo como en el caso anterior y ahora es el estado $|00011\rangle$ que tiene VEE para la evaluación de -5.7 que difiere mucho del valor para el estado óptimo.

Incluyendo la gráfica de la **Figura 4.48** vemos que existe un decremento en la depuración de estados, en otras palabras, que en lugar de comenzar a tener una

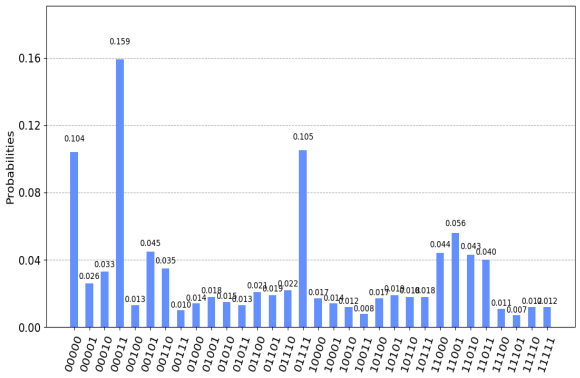


Figura 4.48: Histograma de mejor combinación de parámetros $\gamma = 5.026548\pi$, $\beta_1 = 4.523893\pi$ y $\beta_2 = 0.0\pi$ problema de 5 partículas en configuración completa usando IBM Q.

Estado obtenido	$\|10010\rangle$
VEE	-0.2806
Valor de γ_1	0.0 π
Valor de γ_2	5.026548 π
Valor de β_1	1.005309 π
Valor de β_2	4.775220 π

Tabla 4.43: Resultados de problema ISM con configuración completa de 5 partículas con dos parámetros γ y dos parámetros β usando IBM Q.

menor cantidad de estados (que en concreto sería el estado óptimo y los estados cercanos a este) aún se siguen midiendo estados no deseados, esto contrasta con la **Figura 4.17** donde existe una probabilidad de medición del estado óptimo mucho mayor a la de los otros estados, y además solo se tienen pequeños picos de mediciones hacia ciertos estados en concreto y la mayoría de estados no deseados se presentan con una probabilidad casi nula.

Aumento a dos parejas de parámetros γ_1 y γ_2, con β_1 y β_2

Como ultima experimento se tiene el modelo de QAOA con cuatro parámetros (o dos operadores de fase y dos operadores mezcladores) aplicado al problema de 5 partículas usando los valores: $\gamma_1 = 0.0$, $\gamma_2 = 5.026548$, $\beta_1 = 1.005309$ y $\beta_2 = 4.775220$.

La **Tabla 4.43** contiene los datos para el último modelo probado en IBM Q para el problema de 5 partículas usando BE, en este caso vemos que este modelo nos presenta el peor resultado de todos los modelos simulados para este problema,

en el cual se obtuvo un valor para VEE de -0.2806. Este resultado fue de hecho el "mejor" que se generó usando distintas computadoras cuánticas de IBM Q, con *ibm_santiago* se tuvo un resultado aun más lejano que el presente en la tabla anterior, mientras que con *ibm_manila* se generó este resultado (el de la **Tabla 4.43**) que dentro de todos los realizados para este modelo fue el mejor.

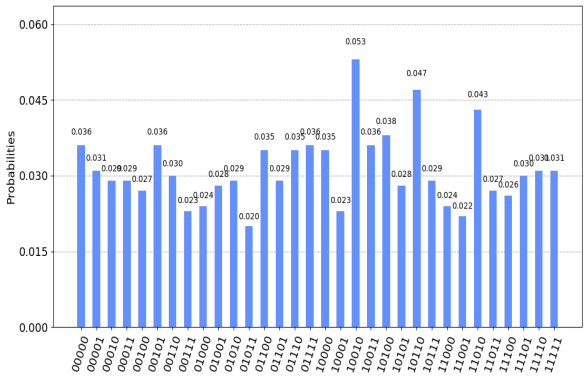


Figura 4.49: Histograma de mejor combinación de parámetros $\gamma_1 = 0.0\pi$, $\gamma_2 = 5.026548\pi$, $\beta_1 = 1.005309\pi$ y $\beta_2 = 4.775220\pi$ problema de 5 partículas en configuración completa usando IBM Q.

Ahora analizando la **Figura 4.49** vemos el porqué de los datos de la tabla anterior, y es que se presenta un histograma con una distribución aleatoria prácticamente uniforme, esto es un resultado bastante contradictorio a lo que se esperaría usando este modelo, ya que la finalidad de aumentar la complejidad del modelo de QAOA es mejorar la respuesta del sistema, aumentando la probabilidad de medición del estado óptimo y sus estados más cercanos, caso contrario a lo ocurrido en esta simulación.

Recopilando los resultados de ISM usando BE en las simulaciones reales, se puede apreciar que existe un deterioro con respecto a los resultados de las simulaciones locales, estos deterioros presentes en los distintos resultados pueden variar dependiendo de muchos factores, sin embargo, los que se pudieron experimentar de primera mano fueron: el tipo de computadora cuántica, donde podemos relacionar el valor de error para compuerta $CNOT$, la última fecha de calibración y el modelo del procesador cuántico que utiliza la computadora; el otro aspecto que se observa se relaciona al estado actual de las computadoras cuánticas (que en su mayoría se consideran de tipo **NISQ**), que al aumentar el número de compuertas cuánticas también se aumenta la relación de error y decoherencias en el sistema, este fenómeno se estudió con mayor detenimiento en los modelos modificados de QAOA (usando dos parámetros para β o aumentando a cuatro parámetros) porque a diferencia de las simulaciones locales que no consideran el ruido dentro de la computadora cuántica y de las imprecisiones que se presentan en las compuertas

Estado obtenido	$\|101\rangle$
VEE	1.5499
Valor de γ	0.502654 π
Valor de β	2.261946 π

Tabla 4.44: Resultados de problema Max-Cut con configuración lineal de 3 nodos usando IBM Q.

cuánticas, en las simulaciones generadas en los dispositivos cuánticos reales se tienen muchos factores que pueden alterar el resultado de manera drástica como en el caso del problema de 5 partículas usando el modelo de QAOA con cuatro parámetros.

Simulaciones usando BE para Max-Cut

Continuando al siguiente bloque de simulaciones se tienen los problemas de Max-Cut usando método de Búsqueda Exhaustiva, estos problemas en teoría deberían tener un mejor resultado que los casos para ISM debido a que estos problemas tienen menos compuertas cuánticas para ser aplicados.

Otro factor que puede favorecer para tener mejores resultados es que estos problemas suelen tener por lo menos dos estados óptimos (un estado óptimo y su inverso), esto puede facilitar la búsqueda de los mejores estados para el sistema.

Simulación usando BE para configuración lineal de 3 nodos

Para el primer problema de Max-Cut usando BE se tiene el de 3 nodos en configuración lineal usando los valores de $\gamma = 0.502654$ y $\beta = 2.261946$.

Los primeros resultados respecto a Max-Cut se tienen en la **Tabla 4.44**, el valor para VEE es de 1.5499 que es un valor bastante cercano al valor esperado para el estado óptimo, el valor de VEE también es similar al obtenido para la simulación local de esta misma instancia. El estado con mayor probabilidad de aparición es uno de los estados óptimos que es $|101\rangle$.

La **Figura 4.50** nos muestra un histograma bastante similar al que se tiene para la **Figura 4.22**, pero para el caso de la simulación local de nueva forma se tiene una mejor distribución de la probabilidad de medición para los estados óptimos, para la simulación generada usando IBM Q se tiene un total de probabilidad (para ambos estados óptimos) de aproximadamente del 60 % y para la simulación local se tiene casi el 70 %.

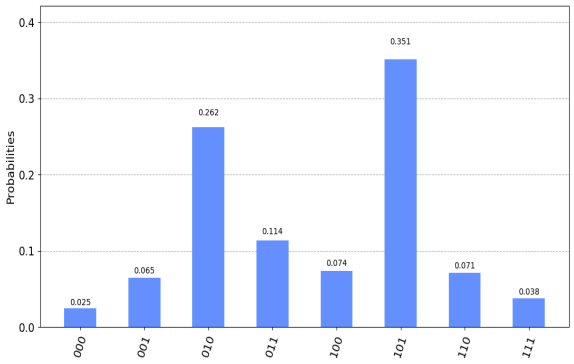


Figura 4.50: Histograma de mejor combinación de parámetros $\gamma = 0.502654\pi$ y $\beta = 2.261946\pi$ problema de 3 nodos en configuración lineal usando IBM Q.

Estado obtenido	$\|1101\rangle$
VEE	1.9819
Valor de γ	1.507964 π
Valor de β	2.764601 π

Tabla 4.45: Resultados de problema Max-Cut con configuración cíclica de 4 nodos usando IBM Q.

Simulación usando BE para configuración cíclica de 4 nodos

Para esta instancia los valores usados para los parámetros para los operadores de fase y mezclador son $\gamma = 1.507964$ y $\beta = 2.764601$.

Los datos de la **Tabla 4.45** nos muestra un valor de VEE de 1.9819 que es muy cercano al resultado de la simulación local para esta instancia, el estado que se presenta con mayor probabilidad (estado obtenido) $|1101\rangle$ es diferente al caso local, y sigue sin ser uno de los estados óptimos.

Al igual que el experimento local de este problema, la **Figura 4.51** nos muestra una distribución prácticamente aleatoria respecto a la medición de los estados, este fenómeno nos indica (al igual que en la simulación local) que no es posible llegar al estado óptimo del sistema usando el modelo actual de QAOA.

Aumento de complejidad de operador mezclador usando dos parámetros β_1 y β_2

Se continúa a experimentar con la modificación del modelo de QAOA con dos parámetros para el operador mezclador aplicado al problema de 4 nodos, los valores usados para los parámetros son $\gamma = 5.780530$, $\beta_1 = 4.775220$ y $\beta_2 =$

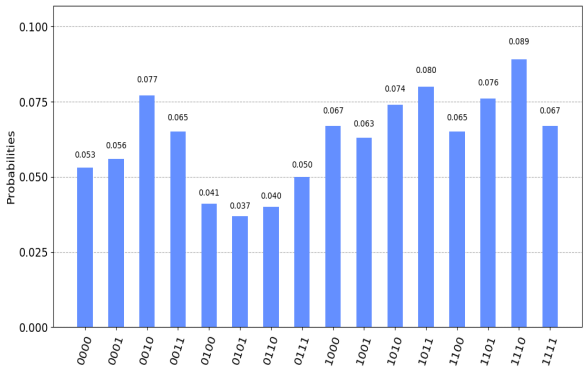


Figura 4.51: Histograma de mejor combinación de parámetros $\gamma = 1.507964\pi$ y $\beta = 2.764601\pi$ problema de 4 nodos en configuración cíclica usando IBM Q.

Estado obtenido	$\|0101\rangle$
VEE	2.312
Valor de γ	5.780530 π
Valor de β_1	4.775220 π
Valor de β_2	1.507964 π

Tabla 4.46: Resultados de problema Max-Cut con configuración cíclica de 4 nodos con dos parámetros β usando IBM Q.

1.507964.

La **Tabla 4.46** nos presenta una mejora en la respuesta con referencia hacia la simulación anterior (usando IBM Q), también se tiene que el estado con mayor probabilidad de aparición es $|0101\rangle$ que es uno de los estados óptimos del sistema. Comparando estos resultados con los de la **Tabla 4.19** vemos que la simulación local presenta un mejor resultado de VEE que se encuentra relativamente cerca del obtenido usando IBM Q, también vemos que en el caso local el estado con mayor probabilidad es el estado inverso a $|0101\rangle$ que es $|1010\rangle$, pero ambos son los estados óptimos del sistema.

El histograma de la **Figura 4.52** también presenta una mejoría con respecto al modelo inicial de QAOA, en este caso vemos que ya existe una pequeña tendencia a tener con mayor probabilidad de medición a los estados óptimos $|0101\rangle$ y $|1010\rangle$; sin embargo, al observarlos junto a los datos generados localmente (**Figura 4.25**) vemos que los resultados locales presentan una tendencia mayor (más probabilidad de aparición) hacia los estados óptimos del sistema.

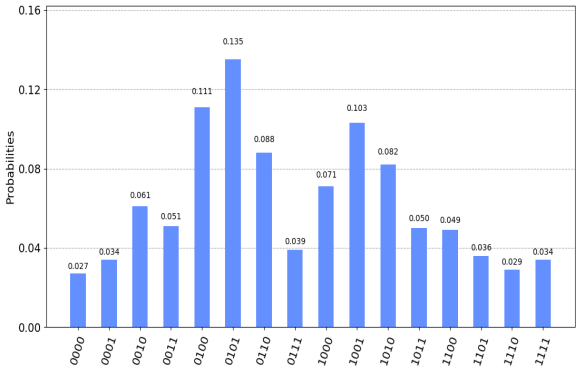


Figura 4.52: Histograma de mejor combinación de parámetros $\gamma = 5.780530\pi$, $\beta_1 = 4.775220\pi$ y $\beta_2 = 1.507964\pi$ problema de 4 nodos en configuración cíclica usando IBM Q.

Estado obtenido	$\|0101\rangle$
VEE	2.8439
Valor de γ_1	1.507964 π
Valor de γ_2	0.753982 π
Valor de β_1	4.775220 π
Valor de β_2	4.775220 π

Tabla 4.47: Resultados de problema Max-Cut con configuración cíclica de 4 nodos con dos parámetros β y dos parámetros γ usando IBM Q.

Aumento a dos parejas de parámetros γ_1 y γ_2, con β_1 y β_2

Modificando el modelo de QAOA para usar cuatro parámetros, dos operadores de fase y dos operadores mezcladores. Los valores usados para este experimento son $\gamma_1 = 1.507964$, $\gamma_2 = 0.753982$, $\beta_1 = 4.775220$ y $\beta_2 = 4.775220$.

Los resultados de la **Tabla 4.47** nos muestran el mejor resultado obtenido para los distintos modelos probados de QAOA, pero en comparación con los datos locales existe una diferencia muy grande entre ambas, siendo la simulación local la que presenta mejores resultados.

La información presente en la **Figura 4.53** expone una mejoría considerable con respecto a los modelos anteriores de QAOA, en este caso el estado óptimo $|0101\rangle$ se presenta con casi el 40 % de probabilidad de aparición, por otro lado, viendo la **Figura 4.26** vemos que la diferencia es muy grande, ya que en la simulación local se tiene casi el 95 % de probabilidad de medición del mismo estado óptimo.

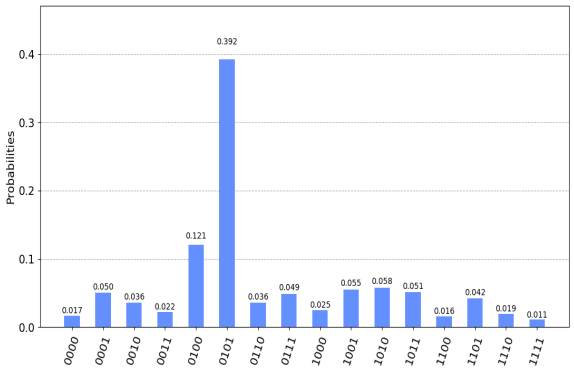


Figura 4.53: Histograma de mejor combinación de parámetros $\gamma_1 = 1.507964\pi$, $\gamma_2 = 0.753982\ \pi$, $\beta_1 = 4.775220\pi$ y $\beta_2 = 4.775220\pi$ problema de 4 nodos en configuración cíclica usando IBM Q.

Estado obtenido	$\|01001\rangle$
VEE	2.9619
Valor de γ	1.256637 π
Valor de β	4.775220 π

Tabla 4.48: Resultados de problema Max-Cut con configuración completa de 5 nodos usando IBM Q.

Simulación usando BE para configuración completa de 5 nodos

BE con QAOA se aplica para el problema de 5 nodos donde se usan los valores $\gamma = 1.256637$ y $\beta = 4.775220$ para los parámetros de los operadores de fase y mezclador.

Los valores contenidos en la **Tabla 4.48** nos dan un valor de VEE de 2.9619, este valor no está cercano al óptimo esperado que es de 6, pero este problema es el que presenta la mayor cantidad de estados óptimos por lo que la prioridad para aumentar el valor de VEE es decrementar la aparición de estados no deseados. Comparando este resultado con el resultado local de la **Tabla 4.21** se puede observa que los datos locales tienen un valor de VEE y un estado con mayor probabilidad de aparición diferente.

Analizando la **Figura 4.54** vemos que se tiene una distribución relativamente similar a la de la **Figura 4.27**, siendo que para este caso (con IBM Q) se tiene una probabilidad total de medición hacia los estados óptimos superior al 60 % que no es un valor alto (con respecto a las simulaciones locales) pero comparado con los resultados de otras instancias de esta sección (problemas de 3 y 4 nodos) es el mejor resultado obtenido hasta el momento en lo que respecta a las simulaciones

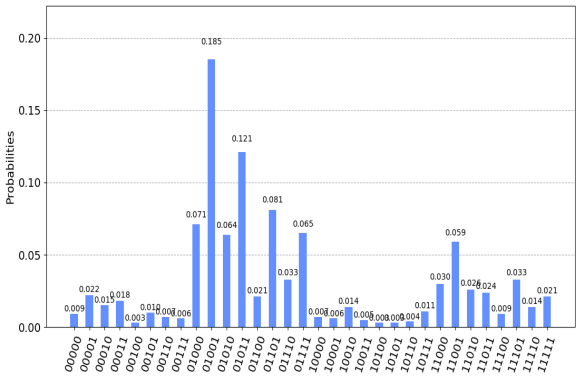


Figura 4.54: Histograma de mejor combinación de parámetros $\gamma = 1.256637\pi$ y $\beta = 4.775220\pi$ problema de 5 nodos en configuración completa usando IBM Q.

Estado obtenido	$\|01101\rangle$
VEE	2.6080
Valor de γ	5.529203 π
Valor de β_1	4.775220 π
Valor de β_2	0.0 π

Tabla 4.49: Resultados de problema Max-Cut con configuración completa de 5 nodos con dos parámetros β usando IBM Q.

en hardware cuántico.

Aumento de complejidad de operador mezclador usando dos parámetros β_1 y β_2

Mejorando el operador mezclador del algoritmo QAOA usando ahora dos parámetros internos con los siguientes valores $\gamma = 5.529203$, $\beta_1 = 4.775220$ y $\beta_2 = 0.0$.

El VEE de la **Tabla 4.49** empeora analizándolo contra el modelo simple de QAOA de la simulación anterior, esto es importante porque el valor de β_2 con 0.0 (que se obtuvo con el método de BE) no aporta rotación alguna dentro de QAOA, sin embargo, debido a que las computadoras cuánticas presentan un grado de distorsión y ruido en el sistema causado por el número de compuertas (entre otros factores), entonces al incrementar las compuertas se tiene un aumento del ruido en el sistema lo que se traduce en una peor respuesta obtenida.

La gráfica que se tiene en la **Figura 4.55** es muy parecida a la obtenida anteriormente, pero con peor respuesta, además si se compara con el resultado de la simulación local también vemos que este resultado no es mejor que el generado

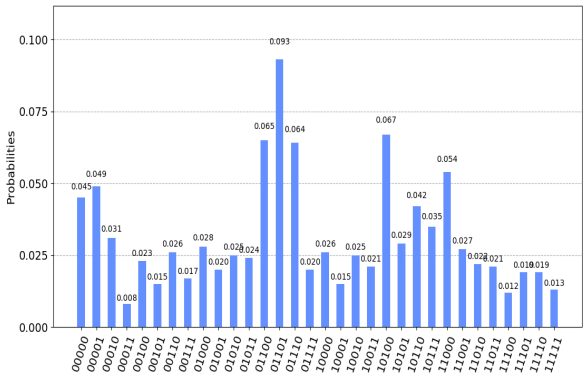


Figura 4.55: Histograma de mejor combinación de parámetros $\gamma = 5.529203\pi$, $\beta_1 = 4.775220\pi$ y $\beta_2 = 0.0\pi$ problema de 5 nodos en configuración completa usando IBM Q.

Estado obtenido	$\|10100\rangle$
VEE	2.6360
Valor de γ_1	0.753982 π
Valor de γ_2	6.031857 π
Valor de β_1	1.507964 π
Valor de β_2	0.0 π

Tabla 4.50: Resultados de problema Max-Cut con configuración completa de 5 nodos con dos parámetros γ y dos parámetros β usando IBM Q.

de los experimentos locales.

Aumento a dos parejas de parámetros γ_1 y γ_2, con β_1 y β_2

Para el último modelo de QAOA con cuatro parámetros se usaron los valores $\gamma_1 = 0.753982$, $\gamma_2 = 6.031857$, $\beta_1 = 1.507964$ y $\beta_2 = 0.0$.

El resultado obtenido para VEE en la **Tabla 4.50** es un valor bastante sorpresivo, ya que comparado con la **Tabla 4.23** (de las simulaciones locales) que nos dio el mejor resultado de los tres modelos, en este caso se tiene prácticamente el mismo resultado del modelo pasado con dos parámetros β, y se menciona además que ambos resultados fueron peores que el primer modelo simple de QAOA (para esta instancia).

Incorporando la **Figura 4.56** al análisis, se observa que efectivamente no hay una mejoría sustancial en la respuesta usando el modelo de cuatro parámetros, esto se pone en contraste con la gráfica de la **Figura 4.29** que presenta un porcentaje de prácticamente el 99 % de medición hacia los estados óptimos, al contrario

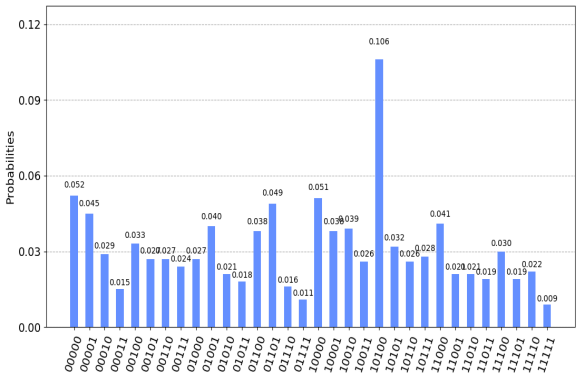


Figura 4.56: Histograma de mejor combinación de parámetros $\gamma_1 = 0.753982\pi$, $\gamma_2 = 6.031857\pi$, $\beta_1 = 1.507964\pi$ y $\beta_2 = 0.0\pi$ problema de 5 nodos en configuración completa usando IBM Q.

de esta simulación usando IBM Q que presenta solo 50 % aproximadamente.

Con este resultado y los anteriores, podemos ver que el fenómeno de falta de precisión y aumento de mediciones erróneas a medida que se aumenta la complejidad del modelo de QAOA (con mayor número de compuertas cuánticas) se repite también en los problemas de Max-Cut. Para todas las simulaciones de este bloque se usó la computadora cuántica *ibmq_manila* que presentó la menor relación de error entre sus compuertas CNOT internas según los datos presentes en IBM Quantum Experience (fecha: 7/10/21). Hasta el momento vemos que existe una tendencia muy marcada referente a la disminución de la calidad de las respuestas de los algoritmos probados en las computadoras cuánticas en comparación con las mismas simulaciones realizadas en una computadora clásica (de manera local), dado que Max-Cut tiene un mayor número de estados óptimos se esperaba que la respuesta fuera mejor que en los casos de ISM porque este fenómeno fue el que se apreció desde las simulaciones locales; sin embargo, se obtuvieron datos relativamente similares (usando optimización por BE) en proporción al error obtenido, donde la mayoría de los modelos más complejos son los que tuvieron una mayor cantidad de errores.

También debe considerarse que muchos modelos presentaron valores nulos de 0.0 para parámetros de QAOA, lo cual genera que esa compuerta (para ese parámetro) no aportará a las rotaciones de los qubits, pero si aportara al aumento el ruido dado el incremento de compuertas del circuito, que en el estado actual de las computadoras cuánticas, las cuales presentan una gran cantidad de imperfecciones que aumentan la generación de resultados aleatorios, ruido y decoherencias al correr el circuito cuántico.

Estado obtenido	$\|001\rangle$
VEE	-2.4612
Valor de γ	1.403145 π
Valor de β	1.480049 π

Tabla 4.51: Resultados de problema ISM con configuración lineal de 3 partículas usando IBM Q.

4.2.2. Algoritmo QAOA usando Búsqueda Local Iterativa

Para la aplicación de BLI se usa la misma técnica que en el caso de BE, donde solo se usarán los datos en referencia a la mejor combinación encontrada por la heurística, esto se hace para respetar el tiempo de procesamiento que se dispone dentro de las computadoras cuánticas de IBM Q.

Simulaciones usando BLI para ISM

El primer grupo de simulaciones reales corresponde los problemas de ISM, los valores usados para los parámetros de QAOA serán los obtenidos por el método BLI en las simulaciones locales al igual que en el caso de BE.

Simulación usando BLI para configuración lineal de 3 partículas

La primera instancia para abordar será la de 3 partículas en configuración lineal usando los valores de $\gamma = 1.403145$ y $\beta = 1.480049$.

Los primeros resultados se pueden ver en la **Tabla 4.51** que presenta un valor de VEE de -2.4612 que está un poco más alejado que en el caso de BE de la **Tabla 4.38** con VEE de -2.64919 y también para la **Tabla 4.24** de la simulación local con VEE de -2.9734.

La **Figura 4.57** nos confirma que el resultado para la mejor combinación de BLI para el caso de 3 partículas tiene valores inferiores a los obtenidos tanto para el caso de BE como el de simulación local usando BLI.

Simulación usando BLI para configuración cíclica de 4 partículas

Pasando a la siguiente instancia del problema para 4 partículas se usaron los valores de $\gamma = 4.933719$ y $\beta = 1.615047$.

Interpretando la información de la **Tabla 4.52** se tiene un valor de VEE de -4.1362 que es un mejor valor comparado con el obtenido para el método de BE (en la **Tabla 4.39**), pero cotejando este valor con el obtenido en la simulación

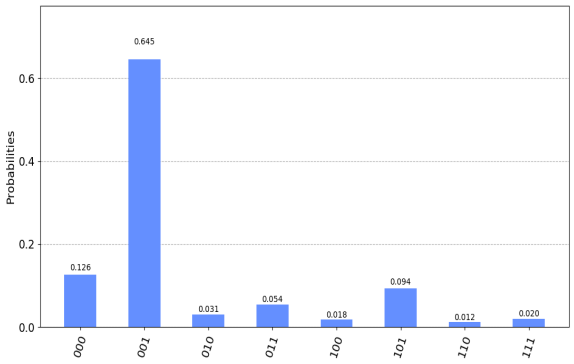


Figura 4.57: Histograma de mejor combinación de parámetros $\gamma = 1.403145\pi$ y $\beta = 1.480049\pi$ del problema de 3 partículas en configuración lineal usando IBM Q.

Estado obtenido	$\|0101\rangle$
VEE	-4.1362
Valor de γ	4.933719 π
Valor de β	1.615047 π

Tabla 4.52: Resultados de problema ISM con configuración cíclica de 4 partículas usando IBM Q.

local tenemos que este valor se encuentra más alejado de la evaluación esperada para el estado óptimo.

Con la mejor combinación probada para esta instancia se genera la **Figura 4.58** que muestra una distribución de probabilidades mucho mejor que para el caso de BE usando IBM Q, donde el estado con mayor probabilidad de aparición es el estado óptimo $|0101\rangle$ con casi el 60 %, además si se compara con BE (usando IBM Q) vemos que también existe una disminución en la probabilidad de aparición de los estados no deseados, la cual genera que el VEE de la tabla anterior sea más cercano al óptimo. En comparación con el mismo experimento, pero ejecutado de manera local vemos que la respuesta local es mejor, ya que el estado óptimo se presenta con un 75 % de probabilidad de medición.

Aumento de complejidad de operador mezclador usando dos parámetros β_1 y β_2

Para la primera modificación de QAOA usando dos parámetros en el operador mezclador se usaron los valores $\gamma = 3.809228$, $\beta_1 = 4.577415$ y $\beta_2 = 3.074400$.

El valor de VEE de la **Tabla 4.53** aparentemente se muestra más lejano

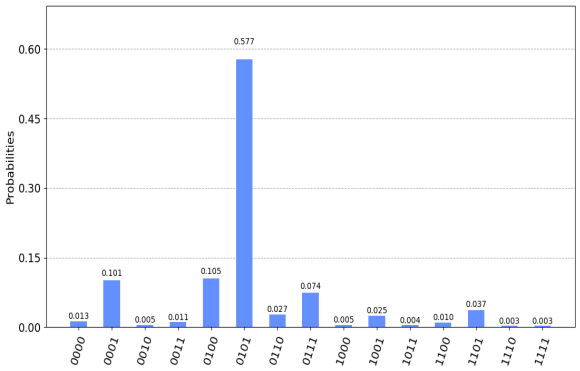


Figura 4.58: Histograma de mejor combinación de parámetros $\gamma = 4.933719\pi$ y $\beta = 1.615047\pi$ del problema de 4 partículas en configuración cíclica usando IBM Q.

Estado obtenido	$\|1101\rangle$
VEE	-3.3936
Valor de γ	3.809228 π
Valor de β_1	4.577415 π
Valor de β_2	3.074400 π

Tabla 4.53: Resultados de problema ISM con configuración cíclica de 4 partículas con dos parámetros β usando IBM Q.

que el valor de VEE para el modelo anterior de QAOA, sin embargo, este fenómeno también se apreció en las simulaciones locales y es que si bien existe un detrimento de VEE con respecto al valor óptimo esperado, existe un aumento en la probabilidad de medición del estado óptimo $|1101\rangle$ caso contrario al modelo simple de QAOA anterior. Las simulaciones locales siguen mostrando mejores resultados porque el valor de VEE de este modelo (usando β_1 y β_2) está más cerca del óptimo.

Con la **Figura 4.59** podemos ver que efectivamente el estado óptimo es el que presenta una probabilidad más alta de aparición, y que el aumento de VEE (es decir que se aleja del mínimo) se da por el incremento de estados no deseados que tienen un valor distinto del óptimo lo cual modifica el valor total de VEE. Comparando este resultado con el de la **Tabla 4.32** vemos que las simulaciones locales presentan un resultado más adecuado que el de las simulaciones realizadas en IBM Q.

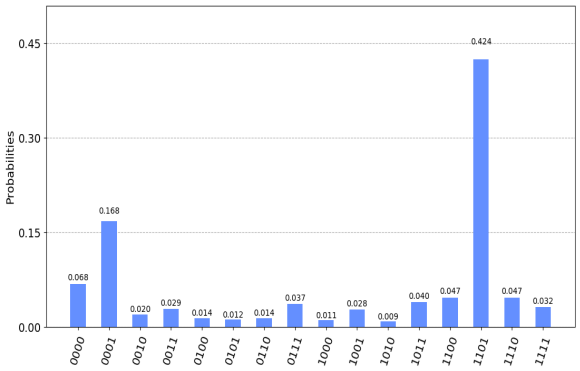


Figura 4.59: Histograma de mejor combinación de parámetros $\gamma = 3.809228\pi$, $\beta_1 = 4.577415\pi$ y $\beta_2 = 3.074400\pi$ del problema de 4 partículas en configuración cíclica usando IBM Q.

Estado obtenido	$\|1101\rangle$
VEE	-1.99739
Valor de γ_1	3.737709 π
Valor de γ_2	5.034290 π
Valor de β_1	4.652072 π
Valor de β_2	3.813496 π

Tabla 4.54: Resultados de problema ISM con configuración cíclica de 4 partículas con dos parámetros β y dos parámetros γ usando IBM Q.

Aumento a dos parejas de parámetros γ_1 y γ_2, con β_1 y β_2

El modelo más complejo de QAOA aplicado para el problema de 4 partículas se realiza con 4 parámetros con valores $\gamma_1 = 3.737709$, $\gamma_2 = 5.034290$, $\beta_1 = 4.652072$ y $\beta_2 = 3.813496$.

La **Tabla 4.54** nos enseña un valor para VEE de -1.99739 que es el valor más lejano del óptimo que se tuvo de los tres modelos probados de QAOA, pero este valor a pesar de estar más lejano aún mantiene el estado óptimo $|1101\rangle$ con mayor probabilidad, si se compara este resultado con su equivalente en las simulaciones locales vemos que la diferencia entre estos es radical, dado que para el caso local el modelo de 4 parámetros fue el mejor, y para el caso real puede verse como un mal resultado en general, e inclusive peor que el modelo simple de QAOA con solo dos parámetros, ya que a pesar de que este modelo no tiene al estado óptimo con mayor probabilidad de medición, el modelo simple se mantiene alejado de los peores estados posibles del sistema.

La información presente en la **Figura 4.60** muestra una disminución en la

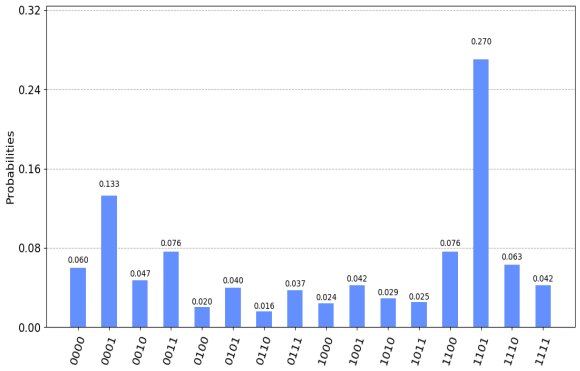


Figura 4.60: Histograma de mejor combinación de parámetros $\gamma_1 = 3.737709\pi$, $\gamma_2 = 5.034290\pi$, $\beta_1 = 4.652072\pi$ y $\beta_2 = 3.813496\pi$ del problema de 4 partículas en configuración cíclica usando IBM Q.

Estado obtenido	$\|00100\rangle$
VEE	0.8364
Valor de γ	5.048866 π
Valor de β	4.685957 π

Tabla 4.55: Resultados de problema ISM con configuración completa de 5 partículas usando IBM Q.

probabilidad de aparición del estado óptimo y un aumento en la probabilidad de estado no deseados como $|1110\rangle$. En comparación con las simulaciones locales, este modelo presenta los peores resultados en simulaciones de IBM Q, caso contrario a la simulación local que presentó los mejores valores.

Simulación usando BLI para configuración completa de 5 partículas

Usando el modelo simple de QAOA con dos parámetros, uno para el operador de fase y uno para el operador mezclador. Los valores usados para estos operadores son $\gamma = 5.048866$ y $\beta = 4.685957$.

Los resultados de la **Tabla 4.55** muestran el peor valor de VEE obtenido para cualquiera de las simulaciones reales, ya que el valor ni siquiera se encuentra dentro de los valores negativos. Esto es bastante curioso porque para el caso local el valor de VEE fue de -8.3734 que se encuentra relativamente cerca del valor óptimo.

Ahora, viendo la **Figura 4.61** vemos que es prácticamente una distribución aleatoria uniforme, donde el estado $|00100\rangle$ que se muestra con mayor probabilidad es uno de los peores estados del sistema. La gráfica para el caso local dispo-

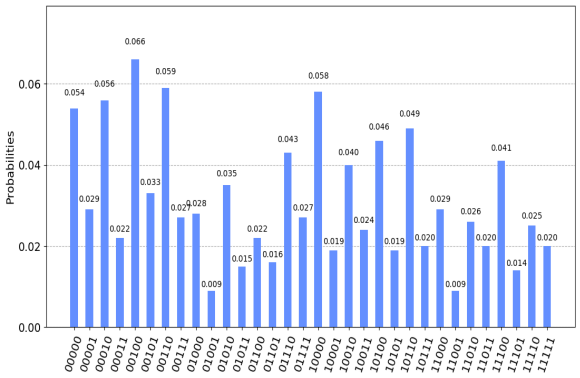


Figura 4.61: Histograma de mejor combinación de parámetros $\gamma = 5.048866\pi$ y $\beta = 4.685957\pi$ del problema de 5 partículas en configuración completa usando IBM Q.

Estado obtenido	$\|01111\rangle$
VEE	-0.6258
Valor de γ	5.031833 π
Valor de β_1	4.710976 π
Valor de β_2	6.245924 π

Tabla 4.56: Resultados de problema ISM con configuración completa de 5 partículas con dos parámetros β usando IBM Q.

nible en la **Figura 4.34** muestra una tendencia hacia el estado óptimo $|00000\rangle$ y presenta otros estados con poca probabilidad, pero dichos estados en su mayoría son estados cercanos al óptimo, caso completamente opuesto a la figura del caso real que no muestra ningún patrón reconocible.

Aumento de complejidad de operador mezclador usando dos parámetros β_1 y β_2

Para el cambio de modelo de QAOA usando dos parámetros β en el operador mezclador se usaron los siguientes valores: $\gamma = 5.031833$, $\beta_1 = 4.710976$ y $\beta_2 = 6.245924$.

Viendo los datos de la **Tabla 4.56** tenemos que existe una ligera mejoría del valor de VEE con respecto del modelo anterior de QAOA, sin embargo, el estado con mayor probabilidad $|01111\rangle$ no es el estado óptimo, y de igual forma si se compara con el VEE de la **Tabla 4.29** vemos que existe una diferencia bastante significativa entre ambos, donde el mejor valor por mucho sigue siendo el de la simulación local.

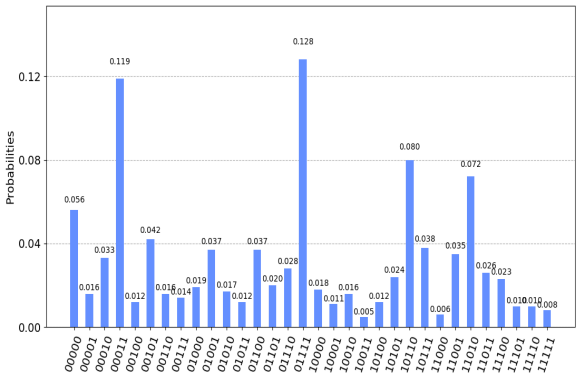


Figura 4.62: Histograma de mejor combinación de parámetros $\gamma = 5.031833\pi$, $\beta_1 = 4.710976\pi$ y $\beta_2 = 6.245924\pi$ del problema de 5 partículas en configuración completa usando IBM Q.

Estado obtenido	$\|11000\rangle$
VEE	0.30479
Valor de γ_1	5.744897 π
Valor de γ_2	0.217068 π
Valor de β_1	1.552818 π
Valor de β_2	3.193465 π

Tabla 4.57: Resultados de problema ISM con configuración completa de 5 partículas con dos parámetros γ y dos parámetros β usando IBM Q.

La gráfica que se muestra en la **Figura 4.62** presenta una ligera mejoría de la distribución de estados comparado con el modelo simple de QAOA (caso anterior), sin embargo, aún existen estados no deseados que aparecen con una probabilidad considerable, esto nos indica porque VEE permanece tan alejado del valor de evaluación para el estado óptimo.

Aumento a dos parejas de parámetros γ_1 y γ_2, con β_1 y β_2

El último modelo probado para el problema de 5 partículas de ISM es usando 4 parámetros con los siguientes valores: $\gamma_1 = 5.744897$, $\gamma_2 = 0.217068$, $\beta_1 = 1.552818$ y $\beta_2 = 3.193465$.

Analizando los datos de la **Tabla 4.57** vemos que ahora VEE empeoró con referencia al modelo anterior de dos parámetros para el operador mezclador, también el estado con más apariciones $|11000\rangle$ no es el estado óptimo. En comparación con los resultados de la **Tabla 4.57** vemos que el caso local aún se mantiene superior (y por mucho) con respecto a la simulación en IBM Q.

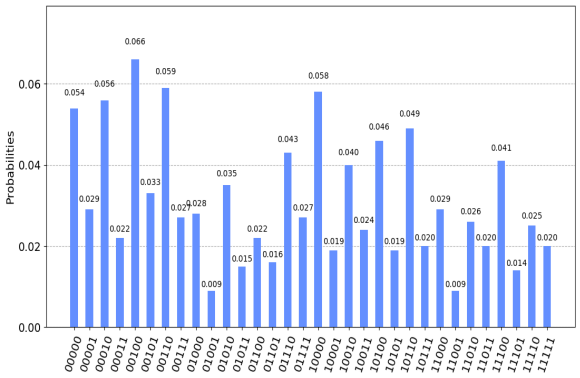


Figura 4.63: Histograma de mejor combinación de parámetros $\gamma_1 = 5.744897\pi$, $\gamma_2 = 0.217068\pi$, $\beta_1 = 1.552818\pi$ y $\beta_2 = 3.193465\pi$ del problema de 5 partículas en configuración completa usando IBM Q.

El histograma de la **Figura 4.63** nos presenta un resultado bastante irregular en cuento la probabilidad de aparición de los estados, donde se aprecia un incremento en estados no deseados que generan que el valor de VEE se aleje aún más del esperado para el estado óptimo. Incorporando la **Figura 4.36** al análisis, no se puede establecer ningún patrón de similitud entre ambos, ya que las probabilidades de aparición para el caso usando IBM Q están muy alejadas de las que se muestran para el caso local, se vuelve a tener el caso local como la mejor solución generada entre ambas simulaciones.

Simulaciones usando BLI para Max-Cut

El último bloque de problemas simulados corresponde a los de tipo Max-Cut usando BLI, de igual forma que en los casos anteriores solo se usarán los valores generados de las simulaciones locales para evitar el uso excesivo de los recursos de IBM Q.

Simulación usando BLI para configuración lineal de 3 nodos

Para esta primera instancia de 3 nodos usando BLI se usan los valores $\gamma = 0.57376$ y $\beta = 4.298174$.

El Valor de Energía Esperado (VEE) que se muestra en la **Tabla 4.58** tiene un valor relativamente cercano al obtenido para el caso local, también el estado que se presenta con mayor probabilidad es el estado óptimo $|010\rangle$.

En la **Figura 4.64** vemos una tendencia a tener con mayor probabilidad los estados óptimos $|010\rangle$ y $|101\rangle$, este patrón es bastante similar al del caso local de

Estado obtenido	$\|010\rangle$
VEE	1.5049
Valor de γ	0.57376 π
Valor de β	4.298174 π

Tabla 4.58: Resultados de problema Max-Cut con configuración lineal de 3 nodos usando IBM Q.

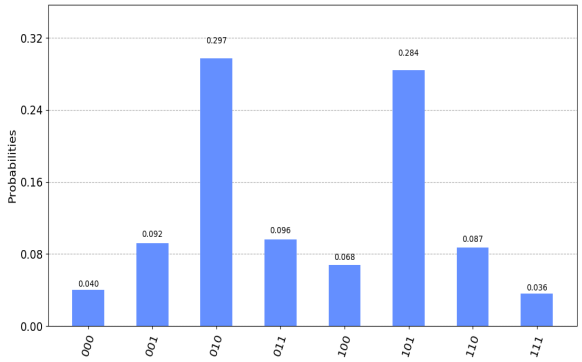


Figura 4.64: Histograma de mejor combinación de parámetros $\gamma = 0.57376\pi$ y $\beta = 4.298174\pi$ del problema de 3 nodos en configuración lineal usando IBM Q.

la **Figura 4.37**, pero al igual que en los casos anteriores, la distribución de la probabilidad entre estados posibles es mejor en la simulación local.

Simulación usando BLI para configuración cíclica de 4 nodos

Transitando al problema de 4 nodos de la simulación real usando BLI se usaron los valores $\gamma = 6.255068$ y $\beta = 0.098935$.

Viendo los datos de la **Tabla 4.59** se aprecia un VEE similar al del caso local, pero con la diferencia de que el estado $|1001\rangle$ se presenta con mayor número de apariciones en el caso de la simulación real.

La distribución de estados de la **Figura 4.65** se muestra con bastante simi-

Estado obtenido	$\|0000\rangle$
VEE	1.9560
Valor de γ	6.255068 π
Valor de β	0.098935 π

Tabla 4.59: Resultados de problema Max-Cut con configuración cíclica de 4 nodos usando IBM Q.

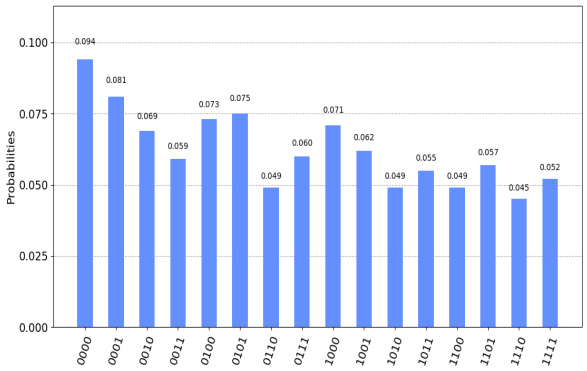


Figura 4.65: Histograma de mejor combinación de parámetros $\gamma = 6.255068\pi$ y $\beta = 0.098935\pi$ del problema de 4 nodos en configuración cíclica usando IBM Q.

Estado obtenido	$\|1011\rangle$
VEE	2.156
Valor de γ	2.698002 π
Valor de β_1	5.458207 π
Valor de β_2	5.488248 π

Tabla 4.60: Resultados de problema Max-Cut con configuración cíclica de 4 nodos con dos parámetros β usando IBM Q.

litud a lo obtenido en ambas simulaciones locales (BE y BLI) y también para el caso de BE en la simulación real, esta distribución es prácticamente unitaria de forma aleatoria sobre todos los estados posibles.

Aumento de complejidad de operador mezclador usando dos parámetros β_1 y β_2

Alterando el modelo simple de QAOA agregando un parámetro más al operador mezclador se obtuvieron los siguientes resultados. Los valores usados para los parámetros fueron $\gamma = 2.698002$, $\beta_1 = 5.458207$ y $\beta_2 = 5.488248$.

Los resultados de la **Tabla 4.60** nos muestra un VEE que se muestra cercano al obtenido en el caso local para el mismo modelo, sin embargo, el estado que se muestra con mayor probabilidad de aparición $|1011\rangle$ no es el estado óptimo.

Observando la **Figura 4.66** se tiene que el patrón de probabilidades de los estados es completamente distinto al del la **Figura 4.39** (caso local) dado que los estados óptimos para el caso de la simulación con IBM Q no se muestran como los de mayor probabilidad de aparición, y para el caso local sí. Vemos que a pesar de tener un VEE cercano, el histograma obtenido es completamente distinto a lo

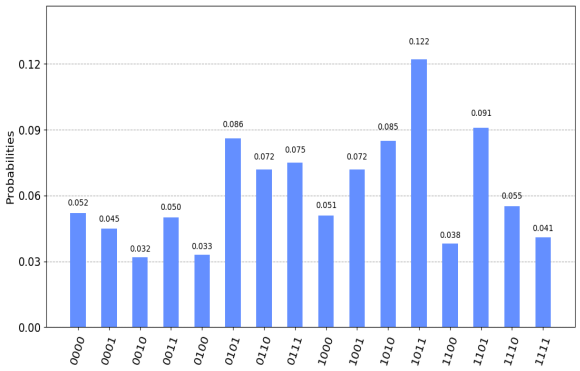


Figura 4.66: Histograma de mejor combinación de parámetros $\gamma = 2.698002\pi$, $\beta_1 = 5.458207\pi$ y $\beta_2 = 5.488248\pi$ del problema de 4 nodos en configuración cíclica usando IBM Q.

Estado obtenido	$\|0101\rangle$
VEE	2.498
Valor de γ_1	1.564337 π
Valor de γ_2	3.939548 π
Valor de β_1	5.315979 π
Valor de β_2	2.370957 π

Tabla 4.61: Resultados de problema Max-Cut con configuración cíclica de 4 nodos con dos parámetros γ y dos parámetros β usando IBM Q.

esperado.

Aumento a dos parejas de parámetros γ_1 y γ_2, con β_1 y β_2

Pasando al último modelo probado para 4 nodos, en el cual se usaron cuatro parámetros (dos para operadores de fase y dos para operadores mezcladores) con los valores $\gamma_1 = 1.564337$, $\gamma_2 = 3.939548$, $\beta_1 = 5.315979$ y $\beta_2 = 2.370957$.

El valor presentado para VEE en la **Tabla 4.61** es el mejor valor obtenido para cualquiera de los tres modelos probados de QAOA para el problema de 4 nodos (usando BLI), pero comparado con el valor obtenido para el caso local, el valor de VEE es mucho peor en esta simulación.

El histograma de la **Figura 4.67** posiblemente el mejor resultado obtenido hasta el momento en lo que respecta a los modelos de tres o cuatro parámetros para QAOA en las simulaciones reales, ya que si bien esta gráfica está muy lejos del resultado obtenido en la **Figura 4.40** que tuvo un resultado "perfecto" con una probabilidad del 100 %, pero tomando en cuenta la tendencia observada con

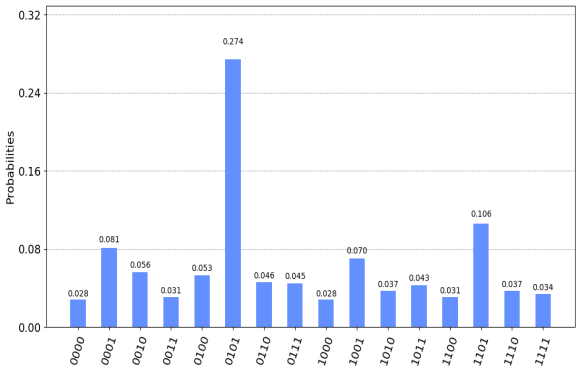


Figura 4.67: Histograma de mejor combinación de parámetros $\gamma_1 = 1.564337\pi$, $\gamma_2 = 3.939548\pi$, $\beta_1 = 5.315979\pi$ y $\beta_2 = 2.370957\pi$ del problema de 4 nodos en configuración cíclica usando IBM Q.

Estado obtenido	$\|10101\rangle$
VEE	2.6859
Valor de γ	4.96846 π
Valor de β	4.724006 π

Tabla 4.62: Resultados de problema Max-Cut con configuración completa de 5 nodos usando IBM Q.

el aumento de compuertas y qubits en el circuito cuántico (haciendo a QAOA más complejo) que genera un aumento de ruido y decoherencias en el sistema, que generaba que las respuestas se encontraran bastante alejadas de las respuestas locales, y en este caso se pudo tener un resultado bastante aceptable considerando la cantidad de elementos del circuito cuántico.

Simulación usando BLI para configuración completa de 5 nodos

Pasando a la última instancia analizada, que corresponde problema de 5 nodos en configuración completa empezamos con el modelo simple de QAOA con solo dos parámetros con valores $\gamma = 4.96846$ y $\beta = 4.724006$.

La **Tabla 4.62** contiene un VEE bastante lejano al valor de evaluación esperado para los estados óptimos, al analizarlo junto al resultado del caso local vemos que nuestro VEE con 2.6859 se encuentra también alejado del caso local, además se debe mencionar que para ambos casos el estado con mayor probabilidad de aparición es $|10101\rangle$ que es uno de los estados óptimos posibles del sistema.

La distribución de probabilidades de estados de la **Figura 4.68** tienen similitudes al del caso local, ya que para ambos casos se muestra que existen proba-

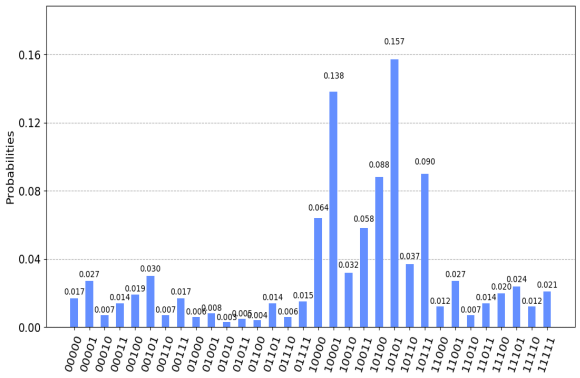


Figura 4.68: Histograma de mejor combinación de parámetros $\gamma = 4.96846\pi$ y $\beta = 4.724006\pi$ del problema de 5 nodos en configuración completa usando IBM Q.

Estado obtenido	$\|11101\rangle$
VEE	2.4840
Valor de γ	5.586943 π
Valor de β_1	4.693765 π
Valor de β_2	3.122219 π

Tabla 4.63: Resultados de problema Max-Cut con configuración completa de 5 nodos con dos parámetros β usando IBM Q.

bilidades superiores al 5 % para varios de los estados óptimos del sistema, pero a diferencia del caso local, esta gráfica (usando IBM Q) nos muestra un aumento en la probabilidad de aparición de estados no deseados.

Aumento de complejidad de operador mezclador usando dos parámetros β_1 y β_2

Usando el modelo modificado de QAOA con dos parámetros para el operador mezclador y uno para el operador de fase se emplearon los siguientes valores con $\gamma = 5.586943$, $\beta_1 = 4.693765$ y $\beta_2 = 3.122219$.

La **Tabla 4.63** nos muestra una deficiencia en la respuesta con respecto al modelo anterior de QAOA (modelo simple) porque el VEE está más alejado del valor de evaluación óptimo esperado que es de 6. También, si lo comparamos con la simulación local vemos que existe de igual forma una diferencia considerable, ya que el modelo local con dos parámetros β no tiene una mejora sustancial con respecto del modelo simple, pero este modelo (modificado de QAOA) decrementa la probabilidad de aparición de estados no deseados (en la simulación local), y

Estado obtenido	$\|01101\rangle$
VEE	2.8280
Valor de γ_1	3.849325 π
Valor de γ_2	2.737281 π
Valor de β_1	1.601462 π
Valor de β_2	3.103111 π

Tabla 4.64: Resultados de problema Max-Cut con configuración completa de 5 nodos con dos parámetros γ y dos parámetros β usando IBM Q.

en esta simulación se incrementó la aparición de estados no deseados.

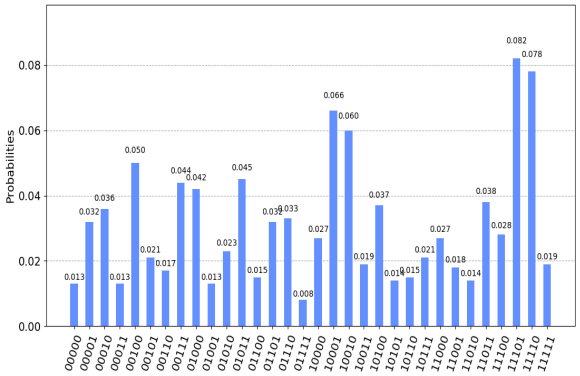


Figura 4.69: Histograma de mejor combinación de parámetros $\gamma = 5.586943\pi$, $\beta_1 = 4.693765\pi$ y $\beta_2 = 3.122219\pi$ del problema de 5 nodos en configuración completa usando IBM Q.

En la **Figura 4.69** se presenta de forma visual los resultados de la tabla anterior, y se puede ver claramente como no existe una mejoría en la respuesta, es decir, en la distribución de las apariciones de los distintos estados del sistema, ya que no se aprecia un aumento en la probabilidad de medición de los estados óptimos sobre los estados no deseados. Estudiando esta figura junto a la **Figura 4.42** vemos que para el caso local si se pudo tener una pequeña mejoría que se aprecia tanto en los estados óptimos como en los no deseados.

Aumento a dos parejas de parámetros γ_1 y γ_2, con β_1 y β_2

La última simulación que se realizará será para el modelo modificado de QAOA usando cuatro parámetros, dos parámetros γ y dos parámetros β, este modelo se aplica al problema de 5 nodos en configuración completa con los siguientes valores para $\gamma_1 = 3.849325$, $\gamma_2 = 2.737281$, $\beta_1 = 1.601462$ y $\beta_2 = 3.103111$.

El valor presentado de VEE con 2.8280 es el mejor resultado obtenido para

cualquiera de los modelos probados en las simulaciones reales (de este problema), aunque no es una mejoría significativa esto nos indica que al menos para esta instancia de 5 nodos y la instancia anterior de 4, el aumento de la complejidad del circuito cuántico que incrementa el ruido y la decoherencia, no afecto tan drásticamente a estos casos, los cuales se vieron beneficiados por una mayor complejidad en el modelo de QAOA, generando una mejor respuesta que en los modelos simples. Comparando estos datos con los de la **Tabla 4.37** vemos que se mantiene la tendencia de que las simulaciones locales poseen mejores resultados independientemente del modelo probado en IBM Q.

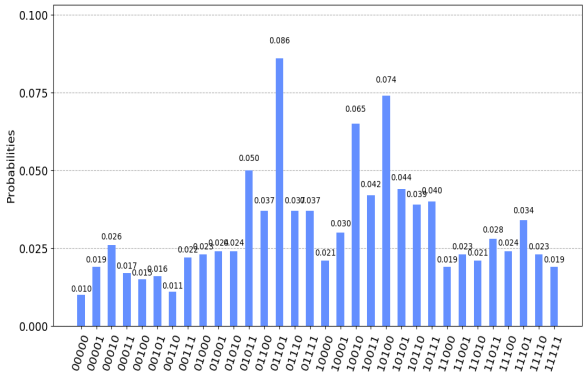


Figura 4.70: Histograma de mejor combinación de parámetros $\gamma_1 = 3.849325\pi$, $\gamma_2 = 2.737281\pi$, $\beta_1 = 1.601462\pi$ y $\beta_2 = 3.103111\pi$ del problema de 5 nodos en configuración completa usando IBM Q.

Finalmente en la gráfica de la **Figura 4.70** vemos una distribución con una pequeña mejoría en torno al número de apariciones de los estados óptimos y un decremento de apariciones en los estados no deseados, aunque no es una diferencia tan marcada como en la **Figura 4.43** que tiene una clara tendencia a medir estados óptimos sobre los otros, si podemos decir que existe un avance con respecto a los modelos anteriores probados en IBM Q para esta instancia.

Anexando estos resultados al análisis podemos ver que de manera general, los problemas de tipo Max-Cut son más compatibles con el uso de QAOA en computadoras cuánticas reales, esto se puede relacionar a dos factores principales; el primero toma en cuenta que para el caso de Max-Cut existe un número inferior de compuertas que son necesarias para plantear el operador de fase (Hamiltoniano del Problema) que aparentemente no se considera como una cantidad elevada, pero a medida que vamos aumentando la complejidad del modelo de QAOA (como en el caso de cuatro parámetros) el aumento de estas compuertas cuánticas extra comienza a incorporar más ruido y al final se traduce en peores respuestas para ISM que para Max-Cut. El otro aspecto que es importante mencionar respecto al desempeño de QAOA con Max-Cut, es que al tener una mayor cantidad de

óptimos globales en el sistema (donde por lo menos existen dos en la mayoría de los casos) esto facilita la exploración por parte de QAOA porque ya no es necesario que encuentre un solo estado óptimo en específico, sino que ahora puede encontrar dos posibles soluciones óptimas dentro del espacio de búsqueda; esto también se puede relacionar con que QAOA no se plantea como un algoritmo exacto (para dar la solución óptima el 100 %) en todos los casos, pero si se busca dar una respuesta aproximada que se refleja como la aparición de un estado óptimo (y los cercanos a este) con una mayor probabilidad.

4.3. Comparativa de resultados y análisis

Con los resultados generados en todas las simulaciones tanto locales como reales, usando tanto el método de optimización por Búsqueda Exhaustiva como el método de Búsqueda Local Iterada (o Iterativa) aplicados para las distintas instancias de los problemas de ISM y Max-Cut usando diferentes modelos de QAOA partiendo del modelo simple (usando solo dos parámetros), el modelo con aumento de parámetros para el operador mezclador y el modelo que utiliza dos operadores de fase y dos operadores mezcladores conectados de manera consecutiva (que representa el modelo más complejo) se generaron las siguientes tablas condensadas para facilitar su análisis y comprensión con miras a las conclusiones de la siguiente sección.

La primera tabla corresponde a la colección de todos los datos de las tablas de simulaciones individuales, donde solo se expresa el VEE (es decir, que las tablas que usaban en método de evaluación por CTA no serán contempladas) y el tipo de simulación al que pertenecen.

La primera parte de la recopilación de datos se muestra en la **Tabla 4.65** donde se tienen todos los valores generados de VEE para los distintos problemas en simulaciones locales y reales, además se agregan dos columnas llamadas ***Dif Opt-Loc*** que corresponde a la diferencia entre el valor óptimo y el valor de simulación local, y ***Dif Opt-Real*** que es la diferencia entre el valor óptimo y el valor de simulación real. Los valores de las columnas de diferencia se busca que estén lo más cercanos a cero posible, para el caso de ISM el signo negativo en la mayoría de los casos se mantiene debido a que representan que tan cerca o lejos se encontraron del valor óptimo que es un número negativo; para el caso de Max-Cut los valores se mantienen normalmente con signo positivo debido a que el óptimo es un número positivo.

La segunda parte de la recopilación que se encuentra en la **Tabla 4.66** muestra los resultados logrados usando la heurística de Búsqueda Local Iterada (o Iterativa) para los distintos problemas analizados, igual que en la tabla anterior, se incorporan dos columnas para agregar las diferencias entre los resultados óptimos y los logrados en las simulaciones locales y reales. La única diferencia que

Búsqueda Exhaustiva					
	ISM para 3 partículas				
	VEE Local	VEE Real	Óptimo	Dif Opt-Loc	Dif Opt-Real
QAOA (2p)	-2.8496	-2.64919	-3.5	***-0.6504***	***-0.85081***
	ISM para 4 partículas				
	VEE Local	VEE Real	Óptimo		
QAOA (2p)	-4.8044	-4.0834	-5.9	***-1.0956***	***-1.8166***
QAOA (3p)	-3.8814	-1.7432	-5.9	***-2.0186***	***-4.1568***
	ISM para 5 partículas				
	VEE Local	VEE Real	Óptimo		
QAOA (2p)	-7.9012	-3.3192	-10.9	***-2.9988***	***-7.5808***
QAOA (3p)	-6.7092	-2.1216	-10.9	***-4.1908***	***-8.7784***
QAOA (4p)	-8.1286	-0.2806	-10.9	***-2.7714***	***-10.6194***
	Max-Cut para 3 nodos				
	VEE Local	VEE Real	Óptimo		
QAOA (2p)	1.658	1.5499	2	***0.342***	***0.4501***
	Max-Cut para 4 nodos				
	VEE Local	VEE Real	Óptimo		
QAOA (2p)	2.088	1.9819	4	***1.912***	***2.0181***
QAOA (3p)	2.618	2.312	4	***1.382***	***1.688***
QAOA (4p)	3.9819	2.8439	4	***0.0181***	***1.1561***
	Max-Cut para 5 nodos				
	VEE Local	VEE Real	Óptimo		
QAOA (2p)	3.514	2.9619	6	***2.486***	***3.0381***
QAOA (3p)	3.634	2.608	6	***2.366***	***3.392***
QAOA (4p)	3.65	2.636	6	***2.35***	***3.364***

Tabla 4.65: Recopilación de resultados para Búsqueda Exhaustiva en los problemas de ISM y Max-Cut para simulaciones locales y reales.

Búsqueda Local Iterada (o Iterativa)					
	ISM para 3 partículas				
	VEE Local	VEE Real	Óptimo	Dif Opt-Loc	Dif Opt-Real
QAOA (2p)	-2.9734	-2.4612	-3.5	***-0.5266***	***-1.0388***
	ISM para 4 partículas				
	VEE Local	VEE Real	Óptimo		
QAOA (2p)	-4.8908	-4.1362	-5.9	***-1.0092***	***-1.7638***
QAOA (3p)	-4.0956	-3.3936	-5.9	***-1.8044***	***-2.5064***
QAOA (4p)	-4.7374	-1.99739	-5.9	***-1.1626***	***-3.90261***
	ISM para 5 partículas				
	VEE Local	VEE Real	Óptimo		
QAOA (2p)	-8.3734	0.8364	-10.9	***-2.5266***	***-11.7364***
QAOA (3p)	-7.3026	-0.6258	-10.9	***-3.5974***	***-10.2742***
QAOA (4p)	-7.3473	0.30479	-10.9	***-3.5527***	***-11.20479***
	Max-Cut para 3 nodos				
	VEE Local	VEE Real	Óptimo		
QAOA (2p)	1.716	1.5049	2	***0.284***	***0.4951***
	Max-Cut para 4 nodos				
	VEE Local	VEE Real	Óptimo		
QAOA (2p)	2.144	1.956	4	***1.856***	***2.044***
QAOA (3p)	2.6919	2.156	4	***1.3081***	***1.844***
QAOA (4p)	4	2.498	4	***0***	***1.502***
	Max-Cut para 5 nodos				
	VEE Local	VEE Real	Óptimo		
QAOA (2p)	3.6879	2.6859	6	***2.3121***	***3.3141***
QAOA (3p)	3.7499	2.484	6	***2.2501***	***3.516***
QAOA (4p)	3.758	2.828	6	***2.242***	***3.172***

Tabla 4.66: Recopilación de resultados para Búsqueda Local Iterada (o Iterativa) en los problemas de ISM y Max-Cut para simulaciones locales y reales.

se encuentra entre ambas tablas es la existencia de una simulación más para el caso de BLI que corresponde al problema de ISM de 4 partículas en configuración cíclica, ya que debido a que BLI permitió incrementar el número de parámetros dentro de QAOA sin aumentar mucho el tiempo y recursos necesarios para la simulación, se analizó que pasaría si para ese problema se agregaba el modelo más complejo de QAOA probado en este trabajo, cosa que no se realizó para el método de BE debido al tiempo y recursos que demandaba.

El patrón que se muestra desde las simulaciones individuales y que ahora se puede confirmar usando las **Tablas 4.65** y **4.66** es que todas las simulaciones locales muestran una mejor respuesta (una menor diferencia entre el valor óptimo y el simulado) que cualquier simulación real, donde las diferencias que existen entre los óptimos y los resultados locales pueden variar dependiendo del problema y del modelo de QAOA usado.

Las diferencias observadas entre los valores óptimos y los de las simulaciones reales es que para el caso de los problemas de ISM a medida que el modelo de QAOA aumenta de complejidad, es decir, aumenta el número de parámetros y esto provoca una diferencia más grande entre el óptimo y el resultado obtenido de la simulación, lo cual nos indica que el ruido y decoherencia que se genera a causa del aumento de compuertas y/o qubits tiene un mayor impacto y que prevalece por encima del beneficio de tener un modelo de QAOA más adecuado.

Ahora, para las diferencias de los tipos de problemas de Max-Cut, se encuentra que en general los resultados tienden a ser un poco mejores a medida que aumenta la complejidad del modelo de QAOA, esto nos muestra que existe una diferencia considerable entre los problemas de tipo Max-Cut e ISM, que aparentemente en la descripción del operador de fase (traducción del problema de optimización al operador cuántico) no existe una "gran" discrepancia con respecto al operador de fase en ISM (existencia del campo magnético externo) pero que en la realidad experimental si afecta considerablemente los resultados, sobre todo para los programas ejecutados en las computadoras cuánticas de IBM Q. Estos resultados también se les puede relacionar con la presencia de más de un óptimo global en los problemas de tipo Max-Cut, ya que QAOA al ser un algoritmo de aproximación de optimización se beneficia del hecho de poder encontrar más de una solución óptima en todo el espacio de búsqueda, lo cual es más "simple" que buscar una única solución en ese mismo espacio.

A partir de estas dos tablas (en específico de las columnas que extraen las diferencias entre óptimos y simulaciones) se generaron dos tablas que expresan el promedio obtenido con respecto a la variación entre valores óptimos y los obtenidos durante las distintas experimentaciones.

Analizando la **Tabla 4.67** se pueden establecer dos resultados importantes entre los métodos de BE y BLI para los casos de las simulaciones locales y reales. El primer resultado se refiere a las simulaciones locales, ya que como podemos ver en la tabla anterior en la columna ***Prom Opt-Loc*** vemos que tanto para

Búsqueda Exhaustiva	
Prom Opt-Loc	*Prom Opt-Real*
ISM	ISM
-2.2876	***-5.63***
Max-Cut	Max-Cut
1.55	***2.16***
Búsqueda Local Iterativa	
Prom Opt-Loc	*Prom Opt-Real*
ISM	ISM
-2.03	***-6.06***
Max-Cut	Max-Cut
1.46	***2.27***

Tabla 4.67: Comparativa de resultados promedio para BE y BLI en los problemas de ISM y Max-Cut para simulaciones locales y reales.

ISM como para Max-Cut se tienen resultados más cercanos al óptimo usando la heurística de Búsqueda Local Iterada. El segundo resultado impacta a las simulaciones reales (columna ***Prom Opt-Real***) y es que para estos casos el método de Búsqueda Exhaustiva se muestra como una mejor estrategia (para ISM y Max-Cut), ya que genera valores más próximos al óptimo (en promedio), esto es un resultado bastante interesante porque se esperaría que la tendencia observada de las simulaciones locales se trasladara a las simulaciones reales cuando se aplica BE o BLI, sin embargo, esto no es así, al menos en los experimentos realizados, pero esto se puede atribuir a muchos factores, ya que como se mostró en las simulaciones reales a lo largo de todas las instancias existía siempre un factor de incertidumbre asociado que no presentaba un patrón reconocible o predecible a lo largo de los experimentos, lo cual puede afectar las lecturas tomadas en los datos estadísticos de las simulaciones reales dependiendo de las computadoras cuánticas usadas y el nivel de calibración de las mismas. También debe considerarse que en el estado actual del hardware cuántico no es posible representar exactamente los valores de las compuertas de rotación planteadas por BLI para los operadores de fase y mezclador, en otras palabras, el valor que se asigna solo se aproxima con una cierta precisión lo cual puede comprometer en gran medida los datos esperados usando la combinación de parámetros hipotética (la obtenida por el método de optimización) y la que se genera en la realidad (la usada realmente por la computadora cuántica), y dado que BE usa valores "estándar" puede que este sea un factor que genere que para las simulaciones reales BE actué de mejor forma, por ello no puede decirse con completa certeza que BE genere mejores valores de parámetros para usarse en QAOA aunque los datos muestren una pequeña mejoría para este método en comparación con BLI. Dado que las simulaciones locales no presentan tantas variaciones en cuanto a los experimentos o la forma en que se ejecutan dichos experimentos, puede tomarse con mayor fiabilidad el

comportamiento observado en los casos locales, es decir, que debe considerarse a BLI como una mejor estrategia para la calibración de los parámetros del algoritmo QAOA mientras no se tenga más información respecto a los fenómenos específicos que alteran los datos vistos en las simulaciones reales.

Capítulo 5

Conclusiones

Las conclusiones que se redactaron para este trabajo se basan en los aspectos más importantes e interesantes que se extrajeron a partir de las distintas simulaciones para las diferentes instancias analizadas tanto de forma local como en las computadoras cuánticas de IBM Quantum Experience.

La primera conclusión se relaciona con las configuraciones y el número de partículas o nodos implementados en las instancias simuladas, ya que los factores que afectan principalmente la complejidad del problema solo serán estos dos aspectos, en este trabajo solo se tomaron problemas con pocos nodos o partículas por lo que no fue posible observar como el aumento de partículas en la simulación (solo del algoritmo QAOA) representa un incremento en los recursos que la computadora clásica necesita para simular el problema, esto se da en el sentido de que la computadora clásica necesita emular el comportamiento cuántico usando aproximaciones que resultan bastante costosas, de hecho esto es uno de los principales problemas por lo que se desarrolló el paradigma de la computación cuántica, ya que a medida que el número de partículas aumenta también la complejidad del problema sube de forma exponencial hasta llegar a un punto en que se vuelve no computable para los paradigmas clásicos. El otro aspecto que puede determinar el desempeño de QAOA (junto con su método de optimización) es la configuración y el tipo de problema, los problemas ISM en su gran mayoría se presentan con un solo estado óptimo independientemente de la configuración de las partículas (lineal, cíclica o completa) pero para el caso de Max-Cut la configuración lineal y la configuración cíclica presentaron una solución dual en la cual se tienen dos estados óptimos y esto facilita que el método de optimización pueda encontrar un estado óptimo (o ambos), y cuando se abordó en específico el problema de Max-Cut en configuración completa se observó que al tener una gran cantidad de estados óptimos esto mejoró el actuar de QAOA, en específico con la heurística de Búsqueda Local Iterativa, ya que la mayoría de los modelos de QAOA nos generaron respuestas con una alta probabilidad de medición hacia los estados óptimos.

La conclusión anterior también la podemos relacionar con problemas con múltiples óptimos globales que son más sencillos para resolver con QAOA, porque

al ser un algoritmo de aproximación el hecho de tener múltiples óptimos permite explotar la "falta" de exactitud y permitir que la probabilidad total de medición está distribuida en los estados óptimos (y los estados más cercanos a estos) como en caso del problema de Max-Cut de 5 nodos en configuración completa.

En términos de la comparativa entre los métodos de optimización de Búsqueda Exhaustiva y Búsqueda Local Iterativa (o Iterada) podemos concluir que en la mayoría de los problemas analizados, para las simulaciones locales, BLI es una mejor alternativa que BE, esto lo podemos ver desde dos perspectivas, la primera relaciona la complejidad computacional y recursos que se necesitan para correr BE en comparación con BLI, y la segunda se tiene de los datos estadísticos recabados durante las simulaciones, porque los valores de BLI tanto para ISM como Max-Cut se muestran más cercanos a los óptimos de todas las instancias en promedio.

BLI es un método más eficiente a medida que el número de parámetros incrementa en comparación con BE que aumenta exponencialmente a medida que se suman más parámetros a explorar. Otro aspecto que se necesita analizar con mayor detenimiento es la relación entre el espacio de búsqueda de los parámetros para los operadores de QAOA y el espacio de estados que genera el sistema (a partir de la aplicación de dichos parámetros), es decir, los estados que son medidos; ya que el espacio de búsqueda de parámetros mapea (a partir de una combinación de valores) a un estado dentro del espacio de Hilbert donde están los qubits que representan el problema, esto es bastante complejo e interesante porque nos indica que la optimización que se realiza para obtener el estado óptimo no se efectúa de forma directa a los estados, sino que se aplica al espacio de búsqueda que "interactúa" con espacio de estados del sistema (espacio de Hilbert) y entonces el problema demanda tomar en cuenta las características de ambos espacios para generar una exploración más adecuada.

BE es un método bastante costoso, y para problemas de este estilo, donde existe un rango continuo de valores que pueden tomar los parámetros representa la necesidad de una buena precisión para lo que BE se vuelve poco útil para instancias que necesiten incluso un número reducido de parámetros manteniendo una buena precisión.

BE y BLI son métodos de optimización que difieren mucho entre sí, los resultados para simulaciones locales nos indican una mejoría al aplicar BLI sobre BE simplemente en los resultados generados, pero al aplicar BLI en las simulaciones reales se tuvieron peores resultados que usando BE, como se comentó anteriormente esto se puede deber a muchos factores que ocurren durante la simulación real y afectan de forma directa el resultado final, por ello se toma como referencia las simulaciones locales (hasta no tener una mejor profundidad de información respecto a los fenómenos que afectan en específico las simulaciones reales con BLI) para decir que BLI es un método de optimización más recomendable en su generalidad, que puede ser adecuado a tipos de problemas sin tener que aumentar

mucho su complejidad (y su costo computacional). Es importante señalar que el aumento de parámetros o variables a optimizar en el caso de BLI no representan un crecimiento de costo computacional tan drástico como en BE, ya que en todas las simulaciones realizadas siempre se mantuvieron los mismos valores de calibración que son los que incrementan o decrementan el costo de BLI. Cuando se aplicó BLI en los distintos problemas de este trabajo el costo y tiempo necesario para optimizar los parámetros y simular el circuito cuántico se mantuvo prácticamente igual, aun con el aumento de nuevos parámetros, cosa contraria al método de BE.

También se concluye que es fundamental tener un número adecuado de parámetros (y por ende, un número adecuado de operadores) para lograr explorar de forma eficiente y de la forma más completa posible el espacio de búsqueda de los parámetros. Además, es importante tener en cuenta que al agregar nuevos operadores de forma consecutiva o al incrementar los parámetros del operador mezclador debe considerarse el fenómeno de entrelazamiento porque existen estados que no pueden ser alcanzados solo con rotaciones individuales (aplicación de operadores o compuertas cuánticas no entrelazadas) y es necesario aplicar estas rotaciones de forma entrelazada.

Finalmente, las últimas conclusiones se dan en torno a los resultados obtenidos entre las simulaciones locales y las simulaciones reales, debido a que todas las simulaciones locales presentaron mejores resultados que las reales, aun cuando se implementaron los mismos valores para los parámetros dentro de QAOA en ambos casos. Esto nos habla directamente del estado de las computadoras cuánticas NISQ, que aún se encuentran en una etapa temprana donde existe una gran cantidad de ruido y decoherencia que afecta al sistema y genera resultados completamente diferentes a los esperados en la teoría (simulaciones locales). Existen cuatro factores que se detectaron que presentan un mayor impacto en la calidad de la respuesta generada por parte de QAOA usando IBM Q. El primer factor es el número de compuertas implementadas debido al tipo de problema o a la complejidad del modelo de QAOA, se observa que a medida que el número de compuertas cuánticas aumenta también aumenta el ruido y decoherencias en el sistema; el segundo factor es el número de qubits que se necesitan para el circuito porque al igual que las compuertas al incrementar el número de qubits también se incrementa el ruido del sistema, IBM Q incluso presenta gráficas de la calidad de los qubits del sistema cuántico que se esté usando, donde se tienen qubits con una mayor fiabilidad que otros; el tercer factor se relaciona directamente con la capacidad de la computadora cuántica para poder "crear" (o implementar) el circuito cuántico deseado con exactitud, esto se presenta cuando se busca generar una rotación con un valor muy especifico y que en la práctica posiblemente esa compuerta tenga que ser creada de forma aproximada usando una secuencia de compuertas disponibles para la computadora cuántica, por lo que a mayor detalle de los elementos del circuito posiblemente se tenga más imprecisiones en la implementación que se esté corriendo realmente; y el cuarto factor detecta-

do se presenta con relación a los tipos de modelos de computadoras cuánticas actuales, si bien en su mayoría son catalogadas como NISQ, existen dentro de esta clase muchos tipos de procesadores cuánticos, diferentes tipos de tecnologías de enfriamiento, y también se contempla el mantenimiento que se tiene de cada equipo por lo que en general es importante tener en cuenta el tipo de dispositivo en el que se ejecutará el circuito cuántico en miras de que sea el mejor disponible (mejor tecnología, mejor procesador y manteamiento más reciente) y el que presente una mejor compatibilidad con nuestro problema (que cuente con los recursos necesarios, llámese número de qubits y/o número de compuertas).

Bibliografía

[1] Wittek, P. (2014). *Quantum machine learning: what quantum computing means to data mining*. Academic Press.

[2] Hidary, J. D. (2019). *Quantum Computing: An Applied Approach* (pp. 1-351). Cham: Springer.

[3] Scherer, W. (2019). *Mathematics of Quantum Computing*. Springer International Publishing.

[4] Umesh V. Vazirani (2018). *Quantum Mechanics and Quantum Computation* [MOOC]. Course of Berkeley University, edx.org.

[5] Abhijith, J., Adedoyin, A., Ambrosiano, J., Anisimov, P., Bärtschi, A., Casper, W., ... & Lokhov, A. Y. (2018). Quantum algorithm implementations for beginners. *arXiv e-prints*, arXiv-1804.

[6] Shi, Y. (2002). Both Toffoli and controlled-NOT need little help to do universal quantum computation. *arXiv preprint quant-ph/0205115*.

[7] Boykin, P. O., Mor, T., Pulver, M., Roychowdhury, V., & Vatan, F. (1999, October). On universal and fault-tolerant quantum computing: a novel basis and a new constructive proof of universality for Shor's basis. In *40th Annual Symposium on Foundations of Computer Science (Cat. No. 99CB37039)* (pp. 486-494). IEEE.

[8] Nielsen, M. A., & Chuang, I. L. (2010). *Quantum Computation and Quantum Information*. Cambridge University Press.

[9] Peter, S. (2018). *QAOA: Quantum Approximate Optimization Algorithm*. Lecture on ISCA, Cambridge - MA. Retrivered from https://www.youtube.com/watch?v=HHIWUi3GmdM.

[10] Koch, D., Patel, S., Wessing, L., & Alsing, P. M. (2020). Fundamentals In Quantum Algorithms: A Tutorial Series Using Qiskit Continued. *arXiv preprint arXiv:2008.10647*.

[11] Jordan, S. P. (2008). Fast quantum algorithm for numerical gradient estimation. *Physical review letters*, 95(5), 050501.

[12] Sweke, R., Wilde, F., Meyer, J. J., Schuld, M., Fährmann, P. K., Meynard-Piganeau, B., & Eisert, J. (2020). Stochastic gradient descent for hybrid quantum-classical optimization. *Quantum*, 4, 314.

[13] Harrow, A. W., & Napp, J. C. (2019). Low-depth gradient measurements can improve convergence in variational hybrid quantum-classical algorithms. *Physical Review Letters*, 126(14), 140502.

[14] Alam, M., Ash-Saki, A., & Ghosh, S. (2020, Marzo). Accelerating quantum approximate optimization algorithm using machine learning. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)* (pp. 686-689). IEEE.

[15] Gilyén, A., Arunachalam, S., & Wiebe, N. (2019). Optimizing quantum optimization algorithms via faster quantum gradient computation. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms* (pp. 1425-1444). Society for Industrial and Applied Mathematics.

[16] Yao, J., Bukov, M., & Lin, L. (2020, Agosto). Policy gradient based quantum approximate optimization algorithm. In *Mathematical and Scientific Machine Learning* (pp. 605-634). PMLR.

[17] Yamamoto, N. (2019). On the natural gradient for variational quantum eigensolver. *arXiv preprint arXiv:1909.05074*.

[18] Shaydulin, R., Safro, I., & Larson, J. (2019, September). Multistart methods for quantum approximate optimization. In *2019 IEEE High Performance Extreme Computing Conference (HPEC)* (pp. 1-8). IEEE.

[19] Rebentrost, P., Schuld, M., Wossnig, L., Petruccione, F., & Lloyd, S. (2019). Quantum gradient descent and Newton's method for constrained polynomial optimization. *New Journal of Physics*, 21(7), 073023.

[20] Khairy, S., Shaydulin, R., Cincio, L., Alexeev, Y., & Balaprakash, P. (2020, April). Learning to optimize variational quantum circuits to solve combinatorial problems. In *Proceedings of the AAAI Conference on Artificial Intelligence* (Vol. 34, No. 03, pp. 2367-2375).

[21] Farhi, E., Goldstone, J., & Gutmann, S. (2014). A quantum approximate optimization algorithm. *arXiv preprint arXiv:1411.4028.*

[22] Williams, R. (2013). Improving exhaustive search implies superpolynomial lower bounds. *SIAM Journal on Computing*, 42(3), 1218-1244.

[23] Nievergelt, J. (2000, November). Exhaustive search, combinatorial optimization and enumeration: Exploring the potential of raw computing power. In *International Conference on Current Trends in Theory and Practice of Computer Science* (pp. 18-35). Springer, Berlin, Heidelberg.

[24] Chisholm, M., & Tadepalli, P. (2002, January). Learning decision rules by randomized iterative local search. In *ICML* (pp. 75-82).

[25] Streif, M., & Leib, M. (2019). Comparison of QAOA with quantum and simulated annealing. *arXiv preprint arXiv:1901.01903.*

[26] Shaydulin, R., & Wild, S. M. (2021). Exploiting symmetry reduces the cost of training QAOA. *IEEE Transactions on Quantum Engineering, 2*, 1-9.

[27] Szegedy, M. (2019). What do QAOA energies reveal about graphs?. *arXiv preprint arXiv:1912.12277.*

[28] Bartschi, A., & Eidenbenz, S. (2020, October). Grover mixers for qaoa: Shifting complexity from mixer design to state preparation. In *2020 IEEE International Conference on Quantum Computing and Engineering (QCE)* (pp. 72-82). IEEE.

[29] Lin, C. Y. Y., & Zhu, Y. (2016). Performance of QAOA on typical instances of constraint satisfaction problems with bounded degree. *arXiv preprint arXiv:1601.01744.*

[30] Zhang, Y., Zhang, R., & Potter, A. C. (2021). QED driven QAOA for network-flow optimization. *Quantum, 5*, 510.

[31] Majumdar, R., Madan, D., Bhoumik, D., Vinayagamurthy, D., Raghunathan, S., & Sur-Kolay, S. (2021). Optimizing Ansatz Design in QAOA for Max-cut. *arXiv preprint arXiv:2106.02812.*

[32] Ward, J., Otterbach, J., Crooks, G., Rubin, N., & da Silva, M. (2018). QAOA Performance Benchmarks using Max-Cut. In *APS March Meeting Abstracts* (Vol. 2018, pp. R15-007).

[33] Shaydulin, R., & Alexeev, Y. (2019, October). Evaluating quantum approximate optimization algorithm: A case study. In *2019 tenth international green and sustainable computing conference (IGSC)* (pp. 1-6). IEEE.

[34] Vikstål, P., Grönkvist, M., Svensson, M., Andersson, M., Johansson, G., & Ferrini, G. (2020). Applying the quantum approximate optimization algorithm to the tail-assignment problem. *Physical Review Applied*, 14(3), 034009.

[35] Zhou, L., Wang, S. T., Choi, S., Pichler, H., & Lukin, M. D. (2020). Quantum approximate optimization algorithm: Performance, mechanism, and implementation on near-term devices. *Physical Review X*, 10(2), 021067.

[36] Schumacher, B. 1995. Quantum coding. *Physical Review A*, 51, 2738.

[37] Benioff, P. (1980). The computer as a physical system: A microscopic quantum mechanical Hamiltonian model of computers as represented by Turing machines. *Journal of statistical physics*, 22(5), 563-591.

Apéndice A

Artículo publicado en congreso IEEE ICCI*CC'20

A.1. Referencia del artículo

- Sarmina, B. G., & Khachaturov, G. (2020, September). QPA*: Design of a searching and path planning algorithm for intelligent agents in two dimensions. In *2020 IEEE 19th International Conference on Cognitive Informatics & Cognitive Computing (ICCI* CC)* (pp. 202-209). Disponible en: IEEE. https://ieeexplore.ieee.org/abstract/document/9450270.

A.2. Correo de aceptación

Dear Brian García Sarmina:

Re: 18: QPA*: Design of a searching and path planning algorithm for intelligent agents in two dimensions

We are delighted to inform you that the above submission has been accepted as a regular paper subject to a careful revision based on comments of reviewers as attached. The acceptance rate of regular papers is about 50 % out of all submissions to IEEE ICCI*CC'20. The conference will be organized online during Sept. 26 – 28, 2020 in order to protect participants for potential COVID-19 influence. This year's program will be enriched by five invited keynote speeches. Conference login information and programs will be announced later.

The deadline for camera-ready papers and registrations is on *Monday, Aug. 31*, 2020. Please fill in the attached registration form and send it to one of the following:

a) Domestic participants: Dr. Yiping Duan at yipingduan@mail.tsinghua.edu.cn, Registration Chair

b) International participants: James Xu at: yifan.xu1@ucalgary.ca, Registration assistant

According to IEEE conference policy, each paper must be registered and presented by at least one author before it is included in the IEEE proceedings and recommended for EI index and selected for journal publication.

The following general instructions must be followed in the revision of your paper:

1) At least one keyword of the conference should be included in the title of your paper (see details in CFP);

2) Author names and email addresses must be included below the title of the paper;

3) All camera-ready papers must be in PDF format confirm to the IEEE double-column format (see: http://www.ucalgary.ca/icci_cc/iccicc-20 or search for the IEEE conference paper template);

4) No page number and extra information should be shown on your camera-ready paper;

5) Each camera-ready paper must be uploaded at https://easychair.org/conferences/?conf=ieeeiccicc20 to replace the draft paper as your submitted for review;

6) Copyright forms must be signed and submitted by the first author with the registration form. IEEE requires all authors to provide a transfer of copyright for publishing your paper in IEEE Xplore Digital Library.

Authors may contact the Registration Chair, Dr. Yiping Duan at yipingduan@mail.tsinghua.edu.cn for any question.

We look forward to meeting you in IEEE ICCI*CC'20 at Tsinghua University, Beijing, China, online during Sept. 26-28, 2020.

Yours sincerely,

Program Co-Chairs IEEE ICCI*CC'20

SUBMISSION: 18 TITLE: QPA*: Design of a searching and path planning algorithm for intelligent agents in two dimensions.

A.3. Artículo completo

QPA*: Design of a searching and path planning algorithm for intelligent agents in two dimensions

Brian García Sarmina[1] and Georgii Khachaturov[2]

[1]brian.garsar.6@gmail.com, Student of Maestría en Ciencias de la Computación, Universidad Autónoma Metropolitana Unidad Azcapotzalco

[2]xgeorge@azc.uam.mx, Departamento de Sistemas, Universidad Autónoma Metropolitana Unidad Azcapotzalco

***Abstract*—Searching algorithms and path planning algorithms are in a variety of applications, where the mobile robotics field is one of the most popular. QPA* algorithm attempts to perform this two strategies at the "same time". The algorithm makes use of three strategies to generate a different solution to the exploration and path planning mainstream, using a "proposed version" of a modified A* (star) algorithm, the idea of the Potential Fields Algorithm (to deal with the collision avoidance) applied as a artificial binary field, and an exploration heuristic named "quadrant grid search". The QPA* algorithm is designed to be applied in a search and rescue robot, where the problem of "localization" and "mapping" is consider to be independent of the exploring and path planning algorithm. The "exploration" part of QPA * is intended to overcome the main problem of path planning algorithms, that is the lack of a true "exploration factor" and preserve the efficiency of making a route using the path planning approach of A* algorithm. Finally, we test three different approximations (for the path planning part) in combination with the quadrant grid search, in order to identify the pros and cons of this design.**

***Index Terms*—Intelligent Agents, Artificial Intelligence, Cognitive Robots, Mobile Robots, Exploration Algorithms, Path Planning Algorithms, Search Matrix Grid.**

I. Introduction

In terms of mobile robots, the exploration task can be defined as the way of guiding the mobile robot thought certain map, and use its sensors to get information about the environment [1], the exploration algorithms play a very important role in today mobile robotics, because this robots are capable of generating exploration maps [1] [2], the quality and reliability of this exploration maps is associated to different factors, where some of the most important are: robot's position and the position of the landmarks (relative to the robot's position), the quality of the sensor measurements (and the uncertainty related to them), etc.

Then, the "path planning" algorithms in robotics, try to solve the problem of generating a path two references in a map, usually an start and goal point. The most referenced (at least, in the bibliography consulted) path planning algorithms have an ideal approach, where they do not consider the associated uncertainty due to the robots sensors, actuators, models, etc. [8] In this sense, for the design of "ASTEC" (the autonomous mobile robot taken as reference for this work) the considerations of the associated uncertainty is handled by the localization and mapping algorithm, not by the exploration and path planning algorithm. Also, having an ideal (sometimes deterministic) approach is not always that bad, because the algorithms, in the case of path planning, are much easier to understand and implement, and as Esmaeel Khanmirza, et al., [17] showed, for two dimensional environments, the deterministic path planning algorithms are better than the probabilistic ones.

Dijkstra's Algorithm

Dijkstra's algorithm is a graph search algorithm. This algorithm is the kernel of a great variety of other searching, and path planning algorithms. The algorithm can find the optimal solution for a single source in a non-negative path between nodes, in most cases is used to find the less expensive path [4]. Dijkstra's Algorithm computational cost is mainly based on the number of nodes and the connections between them [3], and that is one of the main problems of this algorithms, because as the number of nodes or connections between those nodes increases also increases the computational time to solve the problem using this method. [5]

The common mathematical expression, used to represent the functionality of the Dijkstra's Algorithm is:

$$d(v) = min\left\{d(u) + c(u,v)\right\} \tag{1}$$

- $d(v)$ is the "active node" that is being explored (and if it is the case is modified).
- $d(u)$ corresponds to the minimum distance from the closest node, coming from a previous node.
- $c(u,v)$ stands for the distance from u to v.

A (star) Algorithm*

A* Algorithm, is one of the main parts of the design of this searching and path planning algorithm. The A* algorithm it is path planning algorithm, it can be used in different configuration spaces, usually topological or metric spaces. [6]

This algorithm uses two different approaches, an heuristic function (mainly Euclidean, Manhattan or Chebyshev distance function) and Dijkstra's Algorithm. Duchon F., et. al, in the article published in 2013 [7], they analyze the efficiency of the algorithm for obtaining a minimum cost path between an start node and a goal node, where the use of the heuristic complements the approach of Dijkstra, this makes, in general,

an improvement of the computational cost. However, they also showed that in large grid exploration matrices (60,000 or more), even this approach need a great quantity of computational resources.

For every element $i(1, 2, 3, ..., n)$ in the front vector, for an instant t, the equation for A* Algorithm is:

$$a_i^t(f) = g_i(f) + d_i(f) \tag{2}$$

Where:

- $g_i(f)$ is the distance or value, calculated using the heuristic part of the algorithm, from a certain front node $i(1, 2, 3, ..., n)$ to the goal m.
- $d_i(f)$ stand for cost of a path from the initial state (start node) to the actual node, visiting certain nodes.
- a_i^t represents the sum of the heuristic and Dijkstra's approaches.

After the calculation for the front nodes values, is established an other similar strategy from Dijkstra, that is the identification of the node with the lowest cost of the front vector and this node generates the "new start" for the next front vector (the lowest cost node defines the exploration path "direction" for the Dijkstra's method).

$$a^{t+1}(f) = min\{a_1(f), a_2(f), a_3(f), ..., a_n(f)\} \tag{3}$$

The Fig (1.1) shows the general functionality of A*, where the main difference with Dijkstra's Algorithm, is the heuristic (based on the Euclidean distance) implementation to modify the front nodes value. The violet circle represents the visited nodes vector, the green circle corresponds to the front nodes vector, the start node (purple) and the goal node (red), the minimum cost path in color yellow, and the nodes that have "white color" represent the obstacles in the grid map.

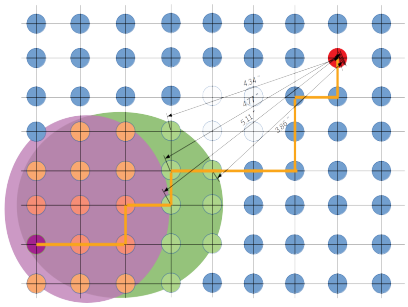


Fig 1.1. A* Algorithm applied to a grid map.

One of the main modifications that A* does to Dijkstra's approach is the "concentration" of the exploring area direction, as the Fig (1.1) shows, for every node in the front vector the Euclidean distance is calculated, this generates that the minimum node or minimum nodes tend to be in the direction of the goal, and this makes the algorithm to be more efficient shrinking the exploration nodes needed to reach the goal.

Potential Fields Algorithm

The Potential Fields Algorithm implements an artificial potential field that obey the Laplace's equation [9], this potential fields are applied in many areas, some applications are: topographic maps, electromagnetic fields, gravitational field, etc. The algorithm tries to regulate the robot and the obstacles in the map, where "regulate" stands for the purpose of avoiding collisions and finding the most effective path to a certain goal.

The artificial potential fields are generated with a function that applies to the obstacles (as a repulsion force) and the goal (as an attraction force). [9] [10]

This kind of algorithms are characterized for being of low computational cost, even when the functions for the map and the obstacles are complex. [10]

The Potential Fields Algorithm, uses a similar strategy to establish the nodes cost as in the case of the A* Algorithm (explained before), where the attractive and repulsive force are dependent of their position relative to the obstacles and goal nodes.

II. QPA* Algorithm Design

The name QPA* came from the three strategies implemented in this design, Q for the Quadrant Grid Search, P for the Potential Fields idea and the A* for the implementation of a modified version of the A * (star) Algorithm.

The search and rescue robot taken as reference, was designed to find "possible victims" in a two dimensional map where the first part of the algorithm is to establish an exploring protocol, this protocol generates pseudo-random "secondary goal" (in the context of the algorithm, a secondary goal is created with the intention of exploring the map), this secondary goal is activated in the absence of an indicator of a possible victim in the map, the indicator is associated with the values of a simulated thermal camera, this values could be pseudo-random simulated or (for this work) are placed manually (one value for the j row and one value for the i column), and if the condition of the indicator is overcome, the exploration problem is done, and the victim's position is establish, and making the path between the actual position of the robot and the location of the possible victim.

Map obstacle generator

For testing purposes, the creation of a map with obstacles is required, the map obstacle generator tries to emulate information that can be obtained with a range sensor (for this work is consider to be an omnidirectional laser sensor LIDAR). The program establish the map of exploration and a set of pseudo-random obstacles in the map. This program also sets an pseudo-random starting point.

The first step is create limits of the exploration map. The limits are generated with the Equation 4 showed below, this equation generates the limits before the implementation of the random obstacles.

$$M_{lim} = \left\{ \begin{matrix} \sum_0^{n-1} \sum_0^{m-1} \ if \ (n = 0 \ or \ n = dim - 1) \\ \sum_0^{n-1} \sum_0^{m-1} \ if \ (m = 0 \ or \ m = dim - 1) \end{matrix} \right\} \tag{4}$$

Where:

- M_{lim} corresponds to the obstacle matrix, that is an $(n \ x \ m)$ matrix, where n and m are the matrix dimensions. In the Equation 4 the matrix goes from 0 to $n - 1$

or $m-1$, this is due to the cyclical conditions in Pyhton syntaxis.

The second step, is the generation of the pseudo-random obstacles in the map, this obstacles are created using two pseudo-random integer values for x and y (the x values are the m elements of the M_{osb} and the y values are the n of the same matrix).

$$Ran_x = randint(0, 40) \tag{5}$$

$$Ran_y = randint(0, 40) \tag{6}$$

The $randint$ function returns a pseudo-random integer value from , in these case, 0 to 40 that are an arbitrary parameters for a possible obstacle size.

$$Osb_x = randint(0, dim - 1) \tag{7}$$

$$Obs_y = randint(0, dim - 1) \tag{8}$$

Then, the Obs_x and Obs_y create the pseudo-random start position of the obstacle creation, where all the obstacles created must be inside the exploration matrix map.

The third step is to add this pseudo-random obstacles created to the matrix M_{obs} that is the pseudo-random obstacles and limits matrix.

$$M^x_{obs} = M^x_{lim} + \left[Obs^x_1, Obs^x_2, Obs^x_3, ..., Obs^x_\rho\right] \tag{9}$$

$$M^y_{obs} = M^y_{lim} + \left[Obs^y_1, Obs^y_2, Obs^y_3, ..., Obs^y_\varphi\right] \tag{10}$$

$$M_{obs} = (M^x_{obs}, M^y_{obs}) \tag{11}$$

Where:

- M^x_{obs} are the position of the obstacles generated in the x axis, that go from 1 to ρ.
- M^y_{obs} are the position of the obstacles generated in the y axis, that go from 1 to φ.
- M_{obs} is the pairs of the position elements in x and y axis.

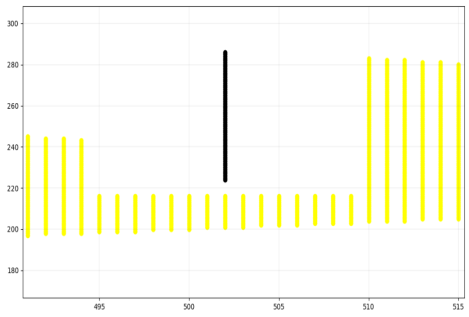


Fig 2.1. Implementation of an artificial field around the obstacles and the map limits.

Now, in Fig (2.1) is implemented the Potential Fields Algorithm idea, the black nodes represent an obstacle, the yellow nodes represent the visited nodes, and the area between this two (the white area) is the "potential field". The illustration shows how an artificial field around the obstacles (and also the limits of the map) is created, it does not properly apply the equations of the Potential Fields Algorithm because it is only consider full or empty permissible nodes for the map (in contrast with Potential Fields, that can apply a spectrum of values to the nodes in the map), instead it used the idea of regulating the robot trajectory by disabling certain areas around the obstacles [9] [10], and with this, can be reduce the collision events in real-world applications for a mobile robot and maintaining a relatively low computational cost (when handling only integer values).

Finally the last step of the map generation program is to apply this artificial field propagation (AFP).

$$M_{final}(x, y) = M_{osb}(x, y) + M_{AFP}(x, y) \tag{12}$$

The $M_{final}(x, y)$ are the final elements of the pseudo-random obstacle exploration matrix, this matrix uses the pseudo-random obstacles generated in (x, y) and the artificial field propagation M_{AFP} elements generated in (x, y) for all the directions allowed.

Implementation of Quadrant Grid Search

The second part of the QPA* algorithm is the quadrant grid search, this heuristic handles the protocol of exploring the map with secondary or primary goals. As it was explained, the exploration protocol consist on establishing secondary goals if the thermal camera indicator does not express a condition of a possible victim.

This part of the algorithm starts with a nine quadrant division of the exploration map is in a positive space for (x, y) where the minimum value is 0 and the maximum is the $(dim - 1)$, the quadrant numbers run from 0 to 8, where the 0 quadrant is in the lower left section and in the upper right section is the eighth 8 quadrant as is shown in the Fig (2.3). This algorithm part is only considered for dimensions that are multiples of 3, like $(3, 6, 9, ..., dim$ x $3)$.

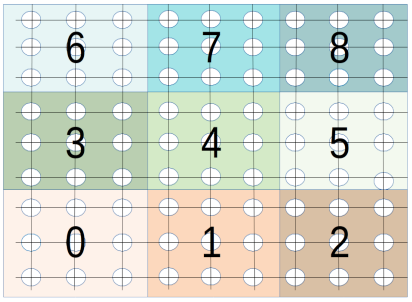


Fig 2.3. Quadrants division for the exploring map.

The steps of the quadrant grid search are listed below.

1) The search and exploring method starts with the pseudo-random start point from the obstacle map generator.
2) If the center of the map $c(x, y) = (\frac{m_{dim}}{2}, \frac{n_{dim}}{2})$ is empty (not obstacles or artificial field), go to the center, else, generate a pseudo-random (empty point) in the fourth 4

quadrant, this is because the fourth quadrant center is equidistant to the others.

3) If the activator (thermal camera) does not indicate a possible victim location, select a quadrant randomly $q_{number} = randint(0, 1, 2, 3, 4, 5, 6, 7, 8)$. If the activator is $true$ ask for the possible victim's location and start the path planning algorithm.
4) If a possible victim was not detected, create a pseudo-random position $q_{pos}(x_r, y_r)$ (in the pseudo-random quadrant selected) and activate the path planning algorithm
5) The secondary goal point position is transformed in the new starting node for the next iteration (if it was the case).
6) Repeat 3, 4 and 5 until the activator indicates the presence of a possible victim.

And an other important consideration to the QPA* algorithm is that: in the generation the pseudo-random secondary goal (new secondary goal position), this goal can not be in the visited nodes vector (generated until that iteration).

Modified A (star) Algorithm*

The final part of this algorithm work, is the path planning part of the algorithm. The path planning strategy is approached with a modified version of an A* Algorithm.

The classic A* Algorithm evaluates the cost of the front nodes using the Equation 3 , and in this work we use the idea of Claus Brenner in the "SLAM Lectures" course [12], where he adds to the classical A* algorithm a Heap Queue Algorithm for the establishing a binary tree of minimum nodes, this Heap Algorithm enhance the search of the minimum of Dijkstra's Algorithm, where "Heaps are binary trees for which every parent node has a value less than or equal to any of its children" [13].

In the following equations k stands for the "level" of the binary tree, and $a[k]$ is consider the parent node.

$$a[k] \leq a[2 * (k + 1)] \in k \tag{13}$$

$$a[k] \leq a[2 * (k + 2)] \in k \tag{14}$$

The use of a heap algorithm it is more efficient than the unsorted search of the minimum proposed in the Equation 3, because the heap algorithm function (for Python 3) inserts the examined node in certain position, where his "children" and "parents" follow the condition of this binary tree. Other interesting property of this algorithm is that the root $k = 0$ is always the minimum, in contrast, the priority queue make a sorted nodes search, which in the case of the modified A* (searching for the minimum element in front) can improve the algorithm performance. [12]

The QPA* algorithm also includes an extra cost parameter related to the visited set of nodes from past iterations $(t - n)$ where n is the number of iterations. This added cost has the purpose of decrease the re-exploration of visited zones (in past iterations). The re-exploration cost is based on the type of connection or movement that connect with one or more previous visited nodes p_{vis}^{t-n}, then, the Equation 2 result as follows.

$$a_i^t(f) = g_i(f) + d_i(f) + p_{vis}^{t-n} \tag{15}$$

This extra value due the previous visited nodes only applies in the exploring protocol of the QPA* algorithm, if the QGS (quadrant grid search) passes an indicator of a possible victim, the extra value of visited nodes is canceled, and only calculates the cost using the Equation 2.

III. Simulation and Result Analysis

In the analysis of our approach we propose to test the quadrant grid search developed with three different versions of the A* Algorithm mentioned before: classical A* (no heap algorithm), Modified A* (with heap algorithm) and the QPA* (with heap algorithm and extra cost for previous visited nodes).

The simulations will consist in two stages, the first stage consists on the graphic simulations where three simulations will be run, and all scenarios will be pseudo-random generated by the map obstacle generator, where the only static parameter is going to be the number of iterations established that is going to be 3 iterations (two secondary goals generated by the quadrant grid search and one primary goal proposed manually).

The second stage, will be similar to the first stage but it will run one hundred simulations per approach (classic A*, modified A* and QPA*), from this simulations we will extract some comparative graphics about the execution time, visited nodes, visited nodes vs. execution time and re-visited nodes.

The evaluation for the graphic stage consists: three iterations (for every simulation) $(0, 1, 2)$, start and goal points in (x, y) format, execution time (seg), grid nodes (are the n x m of the exploration matrix dimensions, for all the simulations the dimensions will be 600 for x and 600 for y), obstacles and artificial field (AF) nodes (this are all the obstacles and artificial fields nodes from the map) and finally, the coverage percentage, the coverage percentage is the number of real visited nodes (all the visited nodes $minus$ the nodes that were re-visited in other iterations) divided by the total of nodes $minus$ the total of obstacles and artificial fields nodes. For the execution time, the Python 3 $time.time()$ [14] function is used, and for counting the distinct "types" of nodes, the function $numpy.count\_nonzero$ [15] that count the non-zero elements in a numpy array.

Also, all the programs generate a (.txt) file where the exploration data is stored for future works.

Simulation for Classic A Algorithm*

For the three runs of the Classic A* Algorithm, the following maps were created, with an starting point (in blue) and a set of obstacles and AFs.

For the first run, the total of obstacles nodes and artificial fields nodes were 30,304 nodes, for the second run the obstacles and AF nodes were 31,391 nodes, and for the third run 32,936 nodes of obstacles and AF nodes.

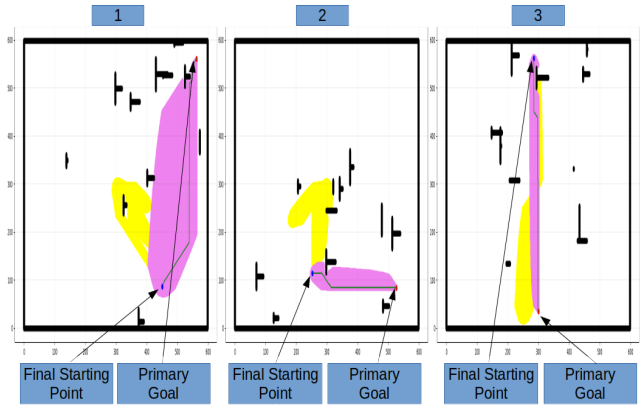


Fig 3.2. Final exploration maps for classic A* Algorithm.

In the Fig (3.2) is exhibit (in yellow color) the visited nodes for secondary goal points (related to the exploration part of the algorithm), in violet color is the final visited nodes for the primary goal (possible victim, setted manually), the blue point is the last starting point (this point it was the last secondary goal of the exploration), the red point is the primary goal location, and the green line is the minimum cost path from the last starting point to the primary goal.

The "start time" (using $time.time()$ function) it was placed after the map obstacle generator for all the algorithm runs (classic A*, modified A* and QPA*), and the "final time" (when the program stops, also, for all the algorithms tested) before the graphic display, also, the $numpy.count\_nonzero$ part is inside the algorithm loop and this can increase the execution time (but it was the same for all the different algorithms tested).

For the first exploration map with classic A*, it was generated in an execution time of 461.65(seg) and a total of visited (explored) nodes of 65,478 (with 2,382 re-visited nodes), for the second exploration map the execution time was 88.32 (seg) and the visited nodes were 20,739 (with 530 re-visited nodes), and for the third exploration map the execution time was 223.46 and the number of visited nodes were 33,565 (with 8,573 re-visited nodes).

Now we can establish the percentage of coverage (of the real visited nodes in relation to the map) and the relation between the total of visited nodes and the execution time.

$$P_{cov} = \frac{(Vis_{real} * 100)}{(M_{total} - Obs_{af})} \tag{16}$$

Where:

- P_{cov} stands for the coverage percentage (for the real visited nodes, not counting the re-visited elements).
- Vis_{real} are all the visited nodes (yellow and violet) $minus$ the re-visited nodes.
- M_{total} all the map grid nodes.
- Obs_{af} are the obstacles nodes and the artificial fields associated to the map.

And for the relation between nodes visited (all visited nodes, including the re-visited elements) and time execution (expressed in units (n/s), where n is the number of visited nodes in one second seg).

$$N_{time} = \frac{Vis_{total}}{E_{time}} \tag{17}$$

Where N_{time} is the number of visited (explored) nodes in one second, Vis_{total} is the total of visited nodes (taking into account the re-visited nodes) and E_{time} is the execution time of the entire exploration (all three iterations for one algorithm run).

Run number	Coverage Percentage	Nodes vs Execution
1	19.137629	141.834723
2	6.149861	234.816576
3	7.560526	150.205853

Table I: Classic A* Algorithm table analysis

In the table I, it is displayed the coverage percentages and the relation, nodes (visited) vs execution (time), the average coverage percentage is 10.949339 % and the average node vs time is 175.619050 (n/s), in particular, the first and third run have a similar node vs time relation, where less number of nodes where visited per second in comparison with the second run, this is due to the fact of the unsorted search that in sometimes could be "fast" (third run) but in general, is very slow, comparing it with a sorted search methods (like Heap algorithm in modified A*). For the re-visited nodes, that can reduce the coverage percentage of the map, the third run was the one that got the highest number with 8,573 re-visited nodes of 33,565 total visited nodes.

Simulation for Modified A Algorithm (using Heap function)*

This modified version of the classic A* Algorithm consists of incorporating a Heap Queue Algorithm (Python 3 function implementation), this algorithm (heap queue algorithm) is also known as priority queue algorithm [13], this modification enhances the performance of a classic A* Algorithm, using the theory behind the heap algorithm that makes the search for the minimum much faster $O(log(F))$ (for the minimum search) and the element insertion in $O(log(F))$, this time complexity compared to an unsorted list (used in the classic A*) is $O(F)$ for the minimum search and the insertion time is $O(1)$.

As Amit Patel says [16] the unsorted lists (implemented in the classical A* approach) have a insertion time complexity of $O(1)$ because usually this method insert the new element in the last position of the array, and for the search for the best the time complexity is $O(F)$ where F is in our case the front vector. For the case of a binary tree (like the Heap Algorithm) the insertion time is $O(log(F))$ and the search for the best is $O(log(F))$ time complexity, this is why the heap function can improve the overall performance of the A* Algorithm.

Now, for the test of the Modified A* Algorithm it was placed in the same circumstances as the Classical A* Algorithm.

For the first map run, the number of obstacles and AF (artificial fields) was 28,641 nodes, for the second run was 29,913 nodes and for the third run 33,206 nodes.

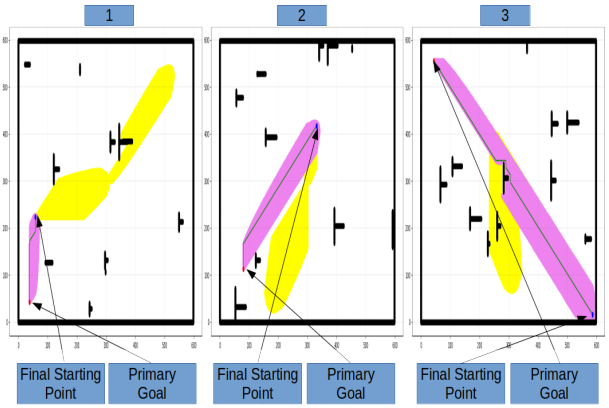


Fig 3.4. Final exploration maps for modified A* Algorithm.

In the first exploration map the total of visited nodes was 35,496 (with 61 re-visited nodes) and the execution time was 35.16 (seg), for the second map, the visited nodes were 45,693 (with 2,010 re-visited nodes) in an execution time of 48.35 (seg) and for the third map 84,192 nodes were visited (with 24,066 re-visited nodes) in an execution time of 39.860556 (seg).

Using the data from the exploration runs and the Equations 16 and 17, the table II was generated.

Run number	Coverage Percentage	Nodes vs Execution
1	10.693839	1009.556313
2	13.233783	945.046535
3	18.398746	1641.169590

Table II: Modified A* Algorithm

The results in the table II come out with an interesting data, where the visited nodes average percentage was 14.108790 % that in compared with the classic A* it is around 4 % more of exploration area covered, the nodes visited vs time execution average is 1198.590813 (n/s) compared with the classic approach (for A* Algorithm) is about seven times more visited (explored nodes) per second, this relation between visited nodes and time execution shows how the modification applying the Heap Queue Algorithm enhances the performance of the classic A* Algorithm approach.

And an other interesting part is the "re-visited nodes", that in the third run it was 24,066 nodes out of 84,192 that were re-visited, and this high number of re-visited nodes shrinks the visited (explored) area by almost a third part.

*Simulation for the QPA**

For the last simulation, it was tested the proposed modified A* Algorithm that implements the Heap Queue Algorithm and a addition of extra cost for previous visited nodes called "QPA* Algorithm". The extra cost is added to the Equation 2 and resulting in the 15, this extra cost is generated by the type of connection or movement that the node is exploring and if this is related to past visited nodes (this value is the same value of the connection between nodes, with an unitary value for the left, right, up and down moves, and a cost of $\sqrt{2}$ for the diagonal moves), the past visited nodes come from previous iterations, where in the first iteration this cost is $zero$ because there is no visited nodes set yet and in the case of a primary goal (when a possible victim is detected) the extra cost value is canceled with the idea of making the algorithm focus on the tentative route for the robot (and not in the exploration part).

The same parameters where used to test the QPA* algorithm.

The number of obstacle nodes and AF nodes for the first map was 30,671, for the second 31,016 and for the third map 32,390 nodes.

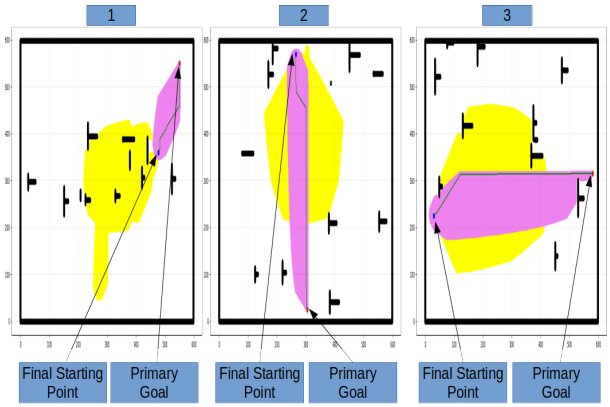


Fig 3.6. Final exploration maps for QPA* Algorithm.

For the first run of the QPA* algorithm, a total of 59,638 nodes were visited (with 2883 re-visited nodes) in an execution time of 44.22 (seg), for the second run 103,047 nodes were visited (with 26,019 re-visited nodes) in an execution time of 63.97 (seg) and for the third run a total of 157,308 were visited (with 46,337 re-visited nodes) in 87.39 (seg).

Now, using the Equations 16 and 17, and the data collected in the three runs, we have:

Run number	Coverage Percentage	Nodes vs Execution
1	17.233526	1348.665762
2	23.413904	1610.864467
3	33.872897	1800.068657

Table III: QPA* Algorithm

Using the table III, the average coverage percentage is 24.840109 % and the average relation between the total visited nodes and the execution time is 1586.532962 (n/s).

The QPA* algorithm was expected to decrease the value of the total re-visited nodes number, but in the simulation the number re-visited nodes was increased in comparison with the other two algorithms, nevertheless, analyzing the results more carefully, this was not completely true, because the largest number of re-visited nodes presented in the first and second algorithm could appear in the three iterations, and in the case of the QPA* the largest number was in the final iteration (the one that corresponds to the possible victim detection) where the protocol of the extra cost for past visited nodes is canceled.

In general, the simple addition of an extra cost for past visited nodes improves the way that the algorithm "explore" using the quadrant grid search approach, for the case of the QPA* algorithm.

Comparative of the three approaches

After the graphical results of the three approaches, this algorithms where tested in one hundred simulations each, where the same conditions as the graphical part were applied, the only difference is the establishment of the primary goal, for the comparative studies the primary goal was setted pseudo-random using a similar approach as in the secondary goal points.

The first comparative is expressed in the Fig 3.7, the figure represents the number of repeated nodes between iterations along the one hundred simulations generated.

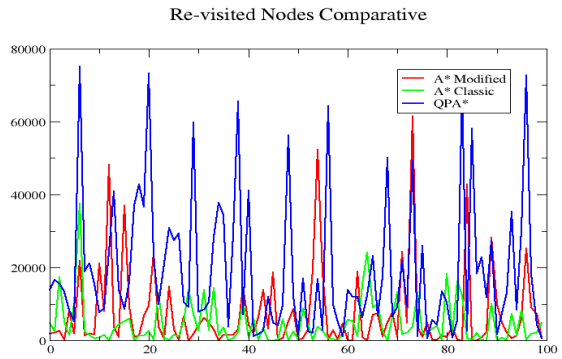


Fig 3.7. Comparative graph of repeated nodes.

In Fig 3.7, the red color represents the "modified A*" algorithm approach, the green color represents the "classic A*" algorithm approach and the blue color express the "QPA*" algorithm approach, in general, the QPA* algorithm generated the highest number of repeated nodes, the modified A* algorithm was the second with highest number of repeated nodes and the approach that generated the least number of repeated nodes between iterations (inside each simulation) was the classic A* algorithm approach.

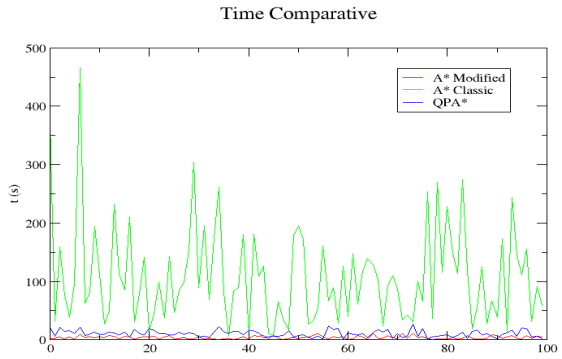


Fig 3.8. Comparative graph of execution time.

Then, in Fig 3.8 the comparative of execution time was made, the results of the modified A* algorithm are in color red, the results of the classic A* algorithm are in color green and the results of the QPA* algorithm are in color blue. The classic A* was the approach that usually take the longest execution time, this information matches with the time complexity associated to the unsorted search of the classic A* algorithm, the modified A* algorithm had the lowest execution time of the three approaches and the QPA* algorithm had a execution time close to the modified A* algorithm, with an increment of 6 seconds in average, this also coincides with the initial purpose of the design of QPA* algorithm, where it preserves the sorted search of the modified A* approach but it presents a small increase in the execution time due to the number of nodes that are explored in the iterations.

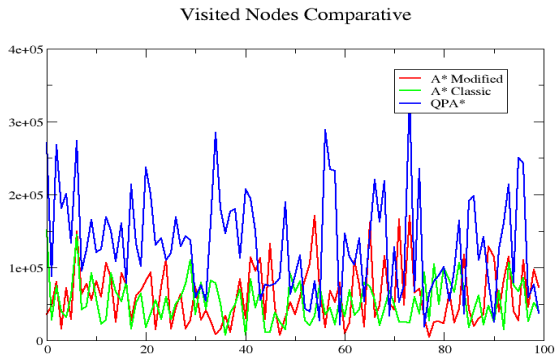


Fig 3.9. Comparative graph of total of visited nodes per simulation.

The Fig 3.9 shows the comparative of the total visited nodes (of every simulation made) for the different approaches, in color red is represented the modified A* algorithm, in color green is the classic A* algorithm and in color blue is the QPA* algorithm. the approach with the tendency to the highest numbers of visited nodes was the QPA* algorithm that had more than double of visited nodes in comparison with the modified A* algorithm that was the second with highest numbers of visited nodes, and the classic A* algorithm had the lowest number of visited nodes.

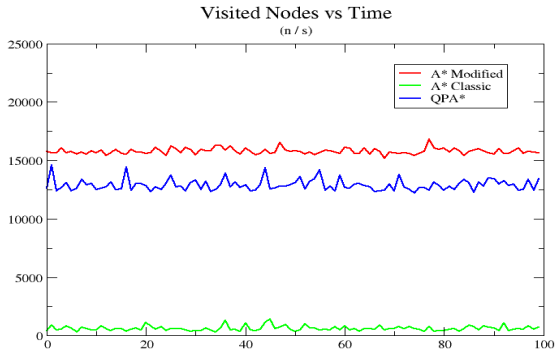


Fig 3.10. Comparative graph of total of visited nodes vs execution time.

Finally in Fig 3.10 we have the total visited nodes vs execution time, where in color red is represented the modified

A* algorithm, in color green is the classic A* algorithm and in color blue is the QPA* algorithm. The approach with better relation visited nodes vs time is the modified A* algorithm due to the execution time that is the lowest of all the approaches presented, the worst relation (n/s) is the classic A* approach because this algorithm had the highest execution time of all, and finally the QPA* had a "good" relation (n/s) considering that the algorithm had a relative "close" execution time (compared with modified A*) and had the highest number of visited nodes.

In theory, the relation between the visited nodes and the area explored by the robot may not seem related, but if we observe it from a practical perspective the path planning part of the algorithm only creates a tentative path for the robot (and in most cases, also a deterministic path), this tentative path is only an "idea" or the "plan" that the robot creates for changing his position from one point to an other. In the case of SLAM approaches, to generate a "real" (more accurate in some cases) localization and map form the robot's perspective, it is necessary to have this "tentative path" as a control input, but for every control command that the robot executes it also generates measurements of the environment, and this measurements are indeed, related to the exploration capacities of the robot. The QPA* algorithm had more visited nodes (compared to the other algorithms), so in the practical sense this approach could improve the amount of new information that can be obtained by a mobile robot. An extra cost to past visited nodes can modified the explored areas of the robot, by making the robot tend to generate routes through not-visited areas (areas without this extra cost), and by using the sensors in this not-visited areas the exploration map increases.

IV. Conclusion

The simulations and analysis shown the enhancement of the classic A* Algorithm by implementing a Heap Queue Algorithm, where it was about seven times more (comparing the classic vs modified version) nodes per second, and almost nine times more for the QPA* (classic vs QPA*) nodes per second. Also the simulations demonstrate that the QPA* map coverage percentage is greater than the classic A* and modified A* algorithms.

Taking the movement or connection values between nodes as the "extra cost", creates a tendency for visiting new areas but not making them completely restricted. The change of the "extra cost" value for other parameter or function, can make QPA* more rigorous (tending not to re-visit past areas). Increasing the cost of re-visiting previous nodes, also increases the execution time for generating the robot's path (as the extra cost increases the QPA* algorithm starts to surround very dramatically the past "visited areas").

Acknowledgment

Thanks to Fis. José de Jesús Cruz Guzmán for all the talks and discussions, and also for the time and patience to teach me many things, always in favor of improving the perspective and understanding of any subject.

Thanks to Professor Zbigniew Oziewicz for the invitation to present this article and also for helping me in reinventing my vision of the sciences, and in general, learning to question everything.

Thanks to Ing. José Luis Barbosa Pacheco, for investing your time, giving me your valuable opinion, knowledge and perspective.

And, thanks to Professor Ańgel López Gómez for the conversations and ideas for implement and improve this work.

References

[1] Cyrill Stachniss, Wolfram Burgard, *Exploring Unknown Environments with Mobile Robots using Coverage Maps*, Published in: IJCAI'03 Proceedings of the 18th international joint conference on Artificial intelligence, August 09-15, 2003, Acapulco - Mxico, San Francisco, CA: Morgan Kaufmann Publishers Inc., 2003.

[2] Robin R. Murphy, *Disaster Robotics*, Cambridge - Massachusetts: MIT Press, 2014.

[3] Mr. S. Julius Fusic, Mr. P. Ramkumar, Dr. K. Hariharan, "Path planning of robot using modified dijkstra Algorithm", in *2018 National Power Engineering Conference (NPEC)*, Madurai - India, 9-10 March 2018, IEEE: 01 October 2018.

[4] E.W. Dijkstra, "A Note on Two Problems in Connexion with Graphs", *Numerische Mathematik*, vol. 1, pp. 269-271, 1959.

[5] M. Barbehenn, "A note on the complexity of Dijkstra's algorithm for graphs with weighted vertices", in *IEEE Transactions on Computers*, vol. 47, no. 2, pp. 263, Feb. 1998.

[6] Frantisek Duchon, Andrej Babinec, Martin Kajan, Peter Beno, Martin Florek, Toms Fico, Ladislav Jurisica, "Path planning with modified A star algorithm for a mobile robot", in *Procedia Engineering, Elsevier*, vol. 96, pp. 56 - 69, Dec. 2014.

[7] Duchon, F., Hunady, D., Dekan, M., Babinec, A., "Optimal navigation for a mobile robot in known environment", in *Applied Mechanicas and Materials*, vol. 282, pp 33 - 38, Jan. 2013.

[8] Sebastian Thrun, Wolfram Burgard, Dieter Fox, *Probabilisti Robotics*, Cambridge, MA: MIT Press, 2006.

[9] F. Haro and M. Torres, "A Comparison of Path Planning Algorithms for Omni-Directional Robots in Dynamic Environments", in *2006 IEEE 3rd Latin American Robotics Symposium (LARS)*, Santiago, 2006, pp. 18-25.

[10] Y. Rasekhipour, A. Khajepour, S. Chen and B. Litkouhi, "A Potential Field-Based Model Predictive Path-Planning Controller for Autonomous Road Vehicles", in *IEEE Transactions on Intelligent Transportation Systems*, vol. 18, no. 5, pp. 1255-1267, May 2017.

[11] Zhanying Zhang and Ziping Zhao, "A Multiple Mobile Robots Path Planning Algorithm Based on A-star and Dijkstra Algorithm", in *International Journal of Smart Home*, Vol. 8, No. 3, 2014, pp. 75 - 86.

[12] Claus Brenner, "SLAM Lectures", 2013, https://www.youtube.com/watch?v=B2qzYCeT9oQ&list=PLpUPoM7Rgzi_7YWn14Va2FODh7LzADBSm

[13] "heapq Heap queue algorithm", Python Software Foundation[US], https://docs.python.org/3/library/heapq.html, consultation date: 25 / Feb / 2019.

[14] "time Time access and conversions", Python Software Foundation[US], https://docs.python.org/3/library/time.html, consultation date: 26 / Feb / 2019.

[15] "numpy.count_nonzero", SciPy.org, https://docs.scipy.org/doc/numpy/reference/generated/numpy.count_nonzero.html, consultation date: 26 / Feb / 2019.

[16] Amit Patel, "Implementation notes: From Amits Thoughts on Pathfinding", Stanfort, http://theory.stanford.edu/~amitp/GameProgramming/ImplementationNotes.html, consultation date: 28 / Feb / 2019.

[17] E. Khanmirza, M. Haghbeigi, M. Nazarahari and S. Doostie, "A Comparative Study of Deterministic and Probabilistic Mobile Robot Path Planning Algorithms", *2017 5th RSI International Conference on Robotics and Mechatronics (ICRoM)*, Tehran, 2017, pp. 534-539.

Apéndice B

Programación

En este apéndice llamado **Programación** se anexan los códigos de referentes a las simulaciones realizadas, estos códigos contienen comentarios para facilitar su lectura y entendimiento, en caso de ser necesario se agregarán algunos textos extra para especificar o resaltar ciertos aspectos del programa.

B.1. Programas para simulaciones locales

Los programas para las simulaciones locales tienen algunas diferencias con respecto a los simulados en hardware cuántico, pero la diferencia más notable es que las simulaciones locales se llevan a cabo en un simulador llamado **qasm_simulator** que es un sistema proporcionado por Qiskit que permite emular los comportamientos cuánticos para obtener resultados estadísticos sobre experimentaciones, similares a si estos hubiesen sido corridos en una computadora cuántica.

B.1.1. Programas usando método de Búsqueda Exhaustiva

El primer bloque de programas corresponde a los usados para el método de optimización por Búsqueda Exhaustiva, este método es bastante simple en su implementación porque puede traducirse como un recorrido (*for loop*) sobre uno o varios ejes de libertad, estos ejes corresponden a las variables o parámetros que se pueden modificar con la finalidad de optimizar el resultado del sistema.

Programas para Ising Spin Model

Las instancias de Modelo de Spin serán las primeras para exponer y explicar el código generado, estas instancias parten desde el problema más simple de 3 partículas hasta el de 5 partículas, donde se toma el primer problema de 3 partículas como base y se van incorporando las modificaciones necesarias para ir adaptando el código a los demás problemas.

Código para problema de 3 partículas en configuración lineal

El primer programa muestra el código generado para las simulaciones del problema de ISM de 3 partículas en configuración lineal, este código es la base para las distintas simulaciones de los problemas de ISM.

```
########-- Problema de 3 particulas para Ising Spin Model --#######
#########################################################################


#- Paqueteria complementaria para ejecucion de QAOA y -#
#- metodo de optimizacion.                            -#
from qiskit import QuantumRegister, ClassicalRegister
from qiskit import QuantumCircuit, Aer, execute
from qiskit.visualization import plot_histogram
import numpy as np
import matplotlib.pyplot as plt
import collections
import math as m


#- Funcion objetivo o funcion de costo -#
def funcion_objetivo(x, h):
    costo = 0
    interaccion = 0
    campo_mag = 0
    # Calculo del costo de interaccion.
    for i in range(len(x)-1):
        if x[i] == x[i+1]:
            interaccion -= 1
        else:
            interaccion += 1
    costo += interaccion
    # Calculo del costo de campo magnetico externo.
    for i in range(len(x)):
        if h[i] > 0:
            if int(x[i]) == 0:
                campo_mag = campo_mag + (-1*h[i])
            else:
                campo_mag = campo_mag + (1*h[i])
        if h[i] < 0:
            if int(x[i]) == 0:
                campo_mag = campo_mag + (-1*h[i])
            else:
                campo_mag = campo_mag + (1*h[i])
    costo += campo_mag
    return costo
#- Funcion objetivo o funcion de costo -#


###############- Busqueda exhaustiva (caso lineal) -###############
def be_3_lineal(corridas, costos, costo_minimo,
                mejores_cuentas, h, num_particulas,
```

```
48                simulador_local, menor_costo_aparicion,
49                minimo):
50
51    for i in np.arange(corridas): # Pruebas para valores de beta.
52
53        # Valor de prueba para "beta" que corresponde al parametro
54        # del operador mezclador.
55        beta = 2*m.pi*(i/corridas)
56
57        for j in np.arange(corridas): # Pruebas para valores
58                                      # de gamma.
59
60            # Valor de prueba para "gamma" que corresponde al
61            # parametro del operador de fase.
62            gamma = 2*m.pi*(j/corridas), 6
63
64            # Creacion de los registros cuanticos "q_reg" y
65            # los registros clasicos "c_reg" para el circuito
66            # cuantico.
67            q_reg = QuantumRegister(num_particulas, name="q_reg")
68            c_reg = ClassicalRegister(num_particulas, name="c_reg")
69            q_c = QuantumCircuit(q_reg,c_reg,name="q_c")
70
71            #* Paso (1)
72            #* El primer paso es generar la superposicion de
73            #* estados iniciales, esto se logra aplicando
74            #* compuertas Hadamard a los qubits del registro
75            #* cuantico.
76            for q in np.arange(num_particulas):
77                q_c.h(q_reg[int(q)]) # Aplicamos la compuerta "h"
78                                    # (Hadamard) a los qubits.
79
80            #* Paso (2)
81            #* Se aplican las compuertas "Z" que corresponden al
82            #* operador de fase, esto se realiza considerando las
83            #* conexiones entre las particulas para el caso del
84            #* Ising Spin Model y tambien se considera el valor de
85            #* "gamma".
86
87            for conexion in np.arange(num_particulas-1):
88                q_c.cx(q_reg[conexion], q_reg[conexion+1])
89                q_c.p(2*gamma, q_reg[conexion+1])
90                q_c.cx(q_reg[conexion], q_reg[conexion+1])
91
92            #* Paso (3)
93            #* Despues de realizar las conexiones entre particulas,
94            #* se procede a agregar el campo magnetico externo "h"
95            #* para cada padricula, esto corresponde al segundo
96            #* termino del operador de fase.
97            for campo in np.arange(num_particulas):
98                q_c.p(gamma*h[int(campo)], campo)
99
100           #* Paso (4)
101           #* Aqui se agrega el operador mezclador, que
```

```
            #* corresponde a los procesos de interferencia
            #* constructiva o destructiva tomando en cuenta el
            #* valor de "beta".
            for mez in np.arange(num_particulas):
                q_c.rx(beta, mez)

            #* Paso (5)
            #* Establecer las mediciones para el registro cuantico,
            #* los valores de los estados base que se obtienen al
            #* medir los qubits (registro cuantico) son almacenados
            #* en los bits clasicos (registro clasico).
            q_c.barrier()
            q_c.measure(q_reg, c_reg)

            # Numero de experimentos que se correrar por cada par
            # de valores gamma y beta.
            n_experimentos = 1000

            # Ejecutar el circuito cuantico en el simulador.
            trabajo = execute(q_c, simulador_local,
                              shots= n_experimentos)

            # Extraer los resultados.
            resultado = trabajo.result()

            # Extraer los datos estadisticos.
            cuentas = resultado.get_counts(q_c)

            #* Aqui se realiza la evaluacion de el costo
            #* dependiendo del estado obtenido, este costo se
            #* multiplica con el numero de apriciones para
            #* obtener el "CTA" (costo total de aparicion).
            for elemento in cuentas.items():
                costo = funcion_objetivo(elemento[0], h)
                costo_aparicion = costo*float(elemento[1])
                if costo_aparicion <= menor_costo_aparicion and:
                    costos.append([costo_aparicion, elemento[0],
                                   elemento[1]])
                    menor_costo_aparicion = costo_aparicion
                    minimo = sorted(costos)
            else:
                continue

            costo_actual = [minimo[0], [gamma, beta]]

            #* Se evalua si el costo actual dada una combinacion de
            #* parametros "gamma" y "beta" es menor con respecto al
            #* "mejor valor actual", ademas, se pregunta
            #* explicitamente si este valor corresponde al estado
            #* optimo del sistema (esto se puede realizar debido a
            #* la tabla de combinaciones generada en el "Cap. 3",
            #* donde este metodo podria ser no recomendable para
            #* problemas donde este estado se desconozca.
            if costo_actual[0][0] < costo_minimo[0][0] and
```

```
    costo_actual[0][1] == '001':
156                 costo_minimo = costo_actual
157                 mejores_cuentas = cuentas
158                 print("Costo actual : ", costo_actual)
159             else:
160                 continue
161 
162             minimo = [[0,0,0], [0.0, 0.0]]
163             costos = [[0,0,0], [0.0, 0.0]]
164 
165     return costo_minimo, mejores_cuentas
166 ################- Busqueda exhaustiva (caso lineal) -################
167 
168 
169 ###-------------------------MAIN-------------------------------###
170 if __name__ == "__main__":
171     # Constantes de calibracion para QAOA.
172     corridas = 100
173     costos = []
174     menor_costo_aparicion = 0
175     minimo =  [[0,0,0], [0.0, 0.0]]
176     costo_minimo = [[0,0,0], [0.0, 0.0]]
177     mejores_cuentas = []
178     h = [0.7, 1.2, -1.6] # Campos magneticos externos por
179                          # particula.
180     num_particulas = 3
181     simulador_local = Aer.backends(name="qasm_simulator")[0]
182     costo_minimo, mejores cuentas = be_3_lineal(corridas, costos,
183                                     costo_minimo, mejores_cuentas,
184                                     h, num_particulas,
185                                     simulador_local, minimo,
186                                     menor_costo_aparicion)
187     print("Costo minimo: ", costo_minimo)
188     plot_histogram(mejores_cuentas)
189     plt.show()
190 ###-------------------------MAIN-------------------------------###
```

Listing B.1: Código para simulación local usando BE en el problema de ISM 3 partículas en configuración lineal.

Agregando unas pequeñas notas a lo que muestra el **Código B.1**, se tiene que el número de *corridas* es el valor que nos dará el número de divisiones en las que se puede separar la gama de valores para γ y β, donde entre más alto sea este número más combinaciones posibles tendrá que explorar el método de BE.

```
1 ########-- Problema de 3 particulas para Ising Spin Model --########
2 ########--                 usando VEE                      --########
3 ######################################################################
4 
5 
6 #- Paqueteria complementaria para ejecucion de QAOA y -#
7 #- metodo de optimizacion.                            -#
```

```
from qiskit import QuantumRegister, ClassicalRegister
from qiskit import QuantumCircuit, Aer, execute
from qiskit.visualization import plot_histogram
import numpy as np
import matplotlib.pyplot as plt
import collections
import math as m

#- Funcion objetivo o funcion de costo -#
def funcion_objetivo(x, h):
    costo = 0
    interaccion = 0
    campo_mag = 0
    # Calculo del costo de interaccion.
    for i in range(len(x)-1):
        if x[i] == x[i+1]:
            interaccion -= 1
        else:
            interaccion += 1
    costo += interaccion
    # Calculo del costo de campo magnetico externo.
    for i in range(len(x)):
        if h[i] > 0:
            if int(x[i]) == 0:
                campo_mag = campo_mag + (-1*h[i])
            else:
                campo_mag = campo_mag + (1*h[i])
        if h[i] < 0:
            if int(x[i]) == 0:
                campo_mag = campo_mag + (-1*h[i])
            else:
                campo_mag = campo_mag + (1*h[i])
    costo += campo_mag
    return costo
#- Funcion objetivo o funcion de costo -#


#############- Busqueda exhaustiva (caso lineal) VEE -###############
def be_3_lineal_vee(corridas, costo_total, costo_actual,
                    costo_minimo, mejores_cuentas, h,
                    num_particulas, simulador_local):

    for i in np.arange(corridas): # Pruebas para valores de beta.

        # Valor de prueba para "beta" que corresponde al parametro
        # del operador mezclador.
        beta = round(2*m.pi*(i/corridas), 6)

        for j in np.arange(corridas): # Pruebas para valores de
                                      # gamma.

            # Valor de prueba para "gamma" que corresponde al
            # parametro del operador de fase.
            gamma = round(2*m.pi*(j/corridas), 6)
```

```
62
63              # Variable para almacenar el "valor esperado".
64              costo_aparicion = 0
65
66              q_reg = QuantumRegister(num_particulas, name="q_reg")
67              c_reg = ClassicalRegister(num_particulas, name="c_reg")
68              q_c = QuantumCircuit(q_reg,c_reg,name="q_c")
69
70              #* Paso (1)
71              #* El primer paso es generar la superposicion de
72              #* estados iniciales, esto se logra aplicando
73              #* compuertas Hadamard a los qubits del registro
74              #* cuantico.
75              for q in np.arange(num_particulas):
76                  q_c.h(q_reg[int(q)]) # Aplicamos la compuerta "h"
77                                      # (Hadamard) a los qubits.
78
79              #* Paso (2)
80              #* Se aplican las compuertas "Z" que corresponden
81              #* al operador de fase, esto se realiza considerando
82              #* las conexiones entre las particulas para el caso
83              #* del Ising Spin Model y tambien se considera el
84              #* valor de"gamma".
85              for conexion in np.arange(num_particulas-1):
86                  q_c.cx(q_reg[conexion], q_reg[conexion+1])
87                  q_c.p(2*gamma, q_reg[conexion+1])
88                  q_c.cx(q_reg[conexion], q_reg[conexion+1])
89
90              #* Paso (3)
91              #* Despues de realizar las conexiones entre
92              #* particulas, se procede a agregar el campo
93              #* magnetico externo "h" para cada padricula, esto
94              #* corresponde al segundo termino del operador de
95              #* fase.
96              for campo in np.arange(num_particulas):
97                  q_c.p(gamma*h[int(campo)], campo)
98
99              #* Paso (4)
100             #* Aqui se agrega el operador mezclador, que
101             #* corresponde a los procesos de interferencia
102             #* constructiva o destructiva tomando en cuenta
103             #* el valor de "beta".
104             for mez in np.arange(num_particulas):
105                 q_c.rx(beta, mez)
106
107             #* Paso (5)
108             #* Establecer las mediciones para el registro
109             #* cuantico, los valores de los estados base que
110             #* se obtienen al medir los qubits (q_reg) son
111             #* almacenados en los bits clasicos (c_reg).
112             q_c.barrier()
113             q_c.measure(q_reg, c_reg)
114
115             # Numero de experimentos que se correrar por cada
```

```
116             # par de valores gamma y beta.
117             n_experimentos = 1000
118 
119             # Ejecutar el circuito cuantico en el simulador.
120             trabajo = execute(q_c, simulador_local,
121                               shots= n_experimentos)
122 
123             # Extraer los resultados.
124             resultado = trabajo.result()
125 
126             # Extraer los datos estadisticos.
127             cuentas = resultado.get_counts(q_c)
128 
129             #* Ahora, a diferencia del metodo de evaluacion
130             #* en el cual se buscaba o preguntaba directamente
131             #* por el estado optimo del sistema, aqui se usa
132             #* una tecnica denominada "VALOR DE ENERGIA
133             #* ESPERADO (VEE) que se usa para conocer el
134             #* aporte total de todos los experimentos del
135             #* circuito para un numero determinado de
136             #* experimentos, usando la correcta combinacion
137             #* de parametros VEE debera tender al valor de la
138             #* funcion costo de evaluada con el estado optimo
139             #* del sistema.
140             for elemento in cuentas.items():
141                 costo = funcion_objetivo(elemento[0], h)
142                 costo_ap = costo*float(elemento[1])
143                 costo_aparicion += (costo_ap / n_experimentos)
144 
145             if costo_aparicion <= costo_actual:
146                 costo_actual = costo_aparicion
147                 costo_minimo = [costo_actual, gamma, beta]
148                 mejores_cuentas = cuentas
149                 print("Costo actual: ", costo_minimo)
150 
151             else:
152                 continue
153 
154     return costo_minimo, mejores_cuentas
155 #############- Busqueda exhaustiva (caso lineal) VEE -###############
156 
157 
158 ###-------------------------MAIN-------------------------------###
159 if __name__ == "__main__":
160     # Constantes de calibracion para QAOA.
161     corridas = 100
162     costo_total = 0
163     costo_actual = 0
164     costo_minimo = [0.0, 0.0, 0.0]
165     mejores_cuentas = []
166     h = [0.7, 1.2, -1.6] # Campos magneticos externos por
167                          # particula.
168     num_particulas = 3
169     simulador_local = Aer.backends(name="qasm_simulator")[0]
```

```
170
171     costo_minimo, mejores_cuentas = be_3_lineal_vee(corridas,
172                                     costo_total, costo_actual,
173                                     costo_minimo, mejores_cuentas,
174                                     h, num_particulas,
175                                     simulador_local)
176     print("Costo minimo: ", costo_minimo)
177     plot_histogram(mejores_cuentas)
178     plt.show()
179 ###-------------------------MAIN-------------------------------###
```

Listing B.2: Código para simulación local usando BE en el problema de ISM 3 partículas en configuración lineal usando VEE.

Ahora para el caso de la aplicación de **Valor de Energía Esperado** (VEE) vemos que el **Código B.2** es prácticamente el mismo, sin embargo, existen variables que no son necesarias en VEE, pero que si son usadas en CTA. El uso de VEE es un método más "realista" debido a que no se debe conocer el estado óptimo de forma previa (el estado que se busca), ya que de manera natural el algoritmo va explorando tratando de encontrar el menor valor de VEE, este método se muestra más eficiente en términos de recursos y tiempo de procesamiento.

Era importante agregar ambos códigos para mostrar la similitud entre ambos, pero ahora para los siguientes códigos (ya que son bastante similares a estos) solo se agregarán las secciones que sean distintas, y obviando la parte del aumento de partículas y otros parámetros de calibración para QAOA dependiendo del tipo de problema abordado.

Código para problema de 4 partículas en configuración cíclica

Para el problema de 4 partículas en configuración cíclica se generó el siguiente código.

```
1  ########-- Problema de 4 particulas para Ising Spin Model --#######
2  ######################################################################
3
4
5  ############################*** CAMBIOS ***############################
6
7  #- Funcion objetivo o funcion de costo -#
8  def funcion_objetivo(x, h):
9      # .
10     # .
11     # .
12     # Calculo del costo de interaccion.
13     for i in range(len(x)):
14         if x[i] == x[i-1]:
15             interaccion -= 1
```

```
        else:
            interaccion += 1
    costo += interaccion
    # .
    # .
    # .
#- Funcion objetivo o funcion de costo -#


###############- Busqueda exhaustiva (caso ciclico) -################
            # .
            # .
            # .
            #* Paso (2)
            #* Se aplican las compuertas "Z" que corresponden al
            #* operador de fase, esto se realiza considerando las
            #* conexiones entre las particulas para el caso del
            #* Ising Spin Model y tambien se considera el valor de
            #* "gamma".
            for conexion in np.arange(num_particulas):
                q_c.cx(q_reg[conexion], q_reg[conexion-1])
                q_c.p(2*gamma, q_reg[conexion])
                q_c.cx(q_reg[conexion], q_reg[conexion-1])
            q_c.barrier()
            # .
            # .
            # .
###############- Busqueda exhaustiva (caso ciclico) -################

############################### CAMBIOS ******************************
```

Listing B.3: Código para simulación local usando BE en el problema de ISM de 4 partículas en configuración cíclica.

En el **Código B.3** se muestran las diferencias entre el programa anterior y este, dado que la configuración cíclica representa una conexión entre la primera y última partícula (para "cerrar" el ciclo), esto genera que tanto la *función de costo* y el *Paso (2)* (que es la primera etapa del operador de fase) tengan que aumentar una interacción más a considerar.

En el problema de 4 partículas también tiene una simulación aumentando el número de variables dentro del operador mezclador, estos cambios generaron las siguientes modificaciones al código base.

```
########-- Problema de 4 particulas para Ising Spin Model --#######
########--                    "2 betas"                      --#######
#####################################################################


############################### CAMBIOS ******************************

```

```
#########- Busqueda exhaustiva (caso ciclico) "2 betas" -##########

    # .
    # .
    # .
    for i in np.arange(corridas): # Pruebas para valores "beta_2".

        # Valor de prueba para "beta_2" que corresponde al segundo
        # parametro del operador mezclador. Este variable permite
        # expandir el espacio de busqueda o el numero de estados
        # que son posibles de alcanzar con el operador mezclador.
        beta_2 = 2*m.pi*(i/corridas)

        for ii in np.arange(corridas): # Pruebas para valores
                                       # "beta".

            # Valor de prueba para "beta" que corresponde al
            # primer parametro del operador mezclador.
            beta = 2*m.pi*(ii/corridas)

            for j in np.arange(corridas): # Pruebas para valores
                                          # "gamma".

                # Valor de prueba para "gamma" que corresponde al
                # parametro del operador de fase.
                gamma = 2*m.pi*(j/corridas)
                # .
                # .
                # .
                ## Paso (4.5) ALTERNATIVO.
                ## Aqui se agrega el segundo parametro para el
                ## operador mezclador, este permite garantizar
                ## una mayor covertura del espacio de busqueda,
                ## que usando un solo parametro. Es importante
                ## mencionar que antes de agregar el nuevo
                ## parametro es necesario generar
                ## entrelazamiento entre las dos variables de
                ## "beta", con miras a poder tambien "alcanzar"
                ## estados con maximo entrelazamiento.
                for conexion in np.arange(num_particulas):
                    q_c.cx(q_reg[conexion], q_reg[conexion-1])
                q_c.barrier()
                for mez2 in np.arange(num_particulas):
                    q_c.rx(beta_2, mez2)
                q_c.barrier()
                # .
                # .
                # .
#########- Busqueda exhaustiva (caso ciclico) "2 betas" -##########

############################## CAMBIOS ******************************
```

Listing B.4: Código para simulación local usando BE en el problema de ISM de 4 partículas (dos βs) en configuración cíclica.

Al agregar una variable extra al operador mezclador llamada β_2 (beta_2) como en el **Código B.4** se agrega un variable más a las posibles combinaciones de parámetros, esto aumenta la complejidad del algoritmo de forma drástica, lo cual a su vez genera un aumento de tiempo de procesamiento y recursos necesarios para simular el problema aun usando un número reducido de divisiones como 25. Otra modificación que se muestra en el código anterior es en el *Paso (4.5)* donde se agregan las compuertas necesarias para agregar este nuevo parámetro, y se incorpora generando un entrelazamiento para garantizar el alcance hacia estados entrelazados.

Para la aplicación de la evaluación con VEE, se realizó una modificación similar a la hecha para el problema de 3 partículas, incorporando también el factor del tipo de configuración y el número de partículas del problema. El modelo también se trasladó al caso de dos βs, donde solo se agrega el parámetro "beta_2" a la variable *costo_minimo*.

Código para problema de 5 partículas en configuración completa

Para la instancia de 5 partículas en configuración completa se generó el siguiente programa.

```
########-- Problema de 5 particulas para Ising Spin Model --#######
#########################################################################


############################### CAMBIOS ****############################

#- Funcion objetivo o funcion de costo -#
def funcion_objetivo(x, h):
    # .
    # .
    # .
    # Calculo del costo de interaccion.
    interacciones = []
    for i in range(len(x)):
        for j in range(len(x)):
            if i != j:
                if [i,j] and [j,i] not in interacciones:
                    if x[i] == x[j]:
                        interaccion -= 1
                        interacciones.append([i,j])
                        interacciones.append([j,i])
                    else:
                        interaccion += 1
                        interacciones.append([i,j])
                        interacciones.append([j,i])
                else:
                    continue
            else:
```

```
                continue
    costo += interaccion
    # .
    # .
    # .
#- Funcion objetivo o funcion de costo -#


##############- Busqueda exhaustiva (caso completo) -################
            # .
            # .
            # .
            #* Paso (2)
            #* Se aplican las compuertas "Z" que corresponden al
            #* operador de fase, esto se realiza considerando las
            #* conexiones entre las particulas para el caso del
            #* Ising Spin Model y tambien se considera el valor de
            #* "gamma".
            conexiones = []
            for conexion in np.arange(num_particulas):
                for conexion_2 in np.arange(num_particulas):
                    if conexion != conexion_2:
                        if [conexion, conexion_2] and [conexion_2,
    conexion] not in conexiones:
                            q_c.cx(q_reg[conexion_2],
                                   q_reg[conexion])
                            q_c.p(2*gamma, q_reg[conexion_2])
                            q_c.cx(q_reg[conexion_2],
                                   q_reg[conexion])
                            conexiones.append([conexion,
                                               conexion_2])
                            conexiones.append([conexion_2,
                                               conexion])
                        else:
                            continue
                    else:
                        continue
            q_c.barrier()
            # .
            # .
            # .
##############- Busqueda exhaustiva (caso completo) -################

############################*** CAMBIOS ***############################
```

Listing B.5: Código para simulación local usando BE en el problema de ISM de 5 partículas (dos βs) en configuración completa.

Revisando las modificaciones que se realizan al **Código B.5** vemos que los cambios se dan en las mismas partes que en caso de 4 partículas, siendo que para la modificación de la *función de costo* ahora se tienen que contemplar todas las conexiones posibles entre las partículas, pero sin generar repeticiones, se anexa

también el cambio en el *Paso (2)* donde se incorporan las compuertas necesarias para proyectar las interacciones entre las partículas (y evitar las repeticiones de estas conexiones).

Para las simulaciones que relacionan al método por VEE, se tienen modificaciones similares a los casos anteriores, tomando en cuenta el número de partículas y conexiones entre partículas.

Para el desarrollo incorporando 2 βs se tiene una aproximación similar al problema de 4 partículas, teniendo en cuenta las características de calibración de este problema.

Finalmente, para el caso del aumento de 2 parámetros β y 2 parámetros γ se realizaron las siguientes modificaciones.

```
########-- Problema de 5 particulas para Ising Spin Model --#######
########--              "2 gammas y 2 betas"              --#######
#########################################################################


############################*** CAMBIOS ***############################

#####- Busqueda exhaustiva (caso completo) 2 betas, 2 gammas -#####
    # .
    # .
    # .
    for i in np.arange(corridas):

        # Segundo valor "beta_2" para segundo operador mezclador.
        beta_2 = 2*m.pi*(i/corridas)

        for ii in np.arange(corridas):

            # Primer valor "beta" para primer operador mezclador.
            beta = 2*m.pi*(ii/corridas)

            for j in np.arange(corridas):

                # Primer valor "gamma" para primer operador de fase
    .
                gamma = 2*m.pi*(j/corridas)

                for jj in np.arange(corridas):

                    # Segundo valor "gamma_2" para segundo operador
                    # de fase.
                    gamma_2 = 2*m.pi*(jj/corridas)
                    # .
                    # .
                    # .
                    ##########################################
                    #* Agregar etapa de entrelazamiento.
                    for ent in np.arange(num_particulas):
```

```
                        q_c.cx(q_reg[ent], q_reg[ent-1])
                    ## Repetir Paso (2), usando "gamma_2".
                    conexiones = []
                    for conexion in np.arange(num_particulas):
                        for conexion_2 in np.arange(num_particulas)
    :
                            if conexion != conexion_2:
                                if [conexion, conexion_2] and [
    conexion_2, conexion] not in conexiones:
                                    q_c.cx(q_reg[conexion_2],
                                           q_reg[conexion])
                                    q_c.p(2*gamma_2,
                                          q_reg[conexion_2])
                                    q_c.cx(q_reg[conexion_2],
                                           q_reg[conexion])
                                    conexiones.append([conexion,
                                                       conexion_2])
                                    conexiones.append([conexion_2,
                                                       conexion])
                                else:
                                    continue
                            else:
                                continue
                    q_c.barrier()
                    ## Repetir Paso (3), usando "gamma_2".
                    for campo in np.arange(num_particulas):
                        q_c.p(gamma_2*h[int(campo)], campo)
                    q_c.barrier()
                    ## Repetir Paso (4), usando "beta_2".
                    for mez in np.arange(num_particulas):
                        q_c.rx(beta_2, mez)
                    q_c.barrier()
                    ##########################################
                    # .
                    # .
                    # .
#####- Busqueda exhaustiva (caso completo) 2 betas, 2 gammas -#####

############################** CAMBIOS **############################
```

Listing B.6: Código para simulación local usando BE en el problema de ISM de 5 partículas (dos βs y dos γs) en configuración completa.

El programa que se aprecia en el **Código B.6** muestra las modificaciones necesarias para incorporar los dos parámetros para los dos operadores de fase y mezcladores (un par de γ y β por par de operadores), existe principalmente un aumento en el número de variables que es necesario explorar en la BE, después se crea una etapa de entrelazamiento para las dos parejas de operadores y se procede a repetir los pasos *(2)*, *(3)* y *(4)* para anexar el segundo par de parámetros γ_2 y β_2. Esto logra ingresar las diferentes etapas (o diferentes operadores, dependiendo del punto de vista) para cada par de operadores de fase y mezcladores, este proceso

se puede repetir más veces de ser necesario, pero debe considerarse el factor del aumento de complejidad debido al número parámetros que es necesario optimizar.

Para el cambio de evaluador con VEE solo debe replicarse lo hecho para los casos anteriores y tener en cuenta el aumento de variables y partículas del problema.

Programas para problemas Max-Cut

La segunda clase de instancias generadas son los problemas de tipo Max-Cut usando BE, estos problemas traducidos a código son bastante similares a los vistos en ISM, la diferencia que existe entre ambos es que para el caso de Max-Cut no se tiene un segundo elemento dentro del operador de fase (aporte de campo magnético externo) porque el planteamiento realizado para estos problemas solo contempla las conexiones entre los nodos, las cuales no tienen peso, dirección o alguna variable extra asociada a estos.

Código para problema de 3 nodos en configuración lineal

El programa usado para simular estos circuitos es muy similar al caso anterior de ISM, la diferencia principal es la extracción de un paso (que sería el *Paso (3)* de ISM), también se agrega el método de evaluación por VEE.

```
##############-- Problema de 3 particulas para Max-Cut --############
##############--              usando VEE                --############
#########################################################################


#- Paqueteria complementaria para ejecucion de QAOA y -#
#- metodo de optimizacion.                            -#
from qiskit import QuantumRegister, ClassicalRegister
from qiskit import QuantumCircuit, Aer, execute
from qiskit.visualization import plot_histogram
import numpy as np
import matplotlib.pyplot as plt
import collections
import math as m

#- Funcion objetivo o funcion de costo -#
def funcion_objetivo(x):
    costo = 0
    interaccion = 0
    # Calculo del costo de interaccion.
    for i in range(len(x)-1):
        if x[i] == x[i+1]:
            interaccion += 0 # No suma conexion.
        else:
            interaccion -= 1 # Suma conexion, se usa signo (-)
```

```
                                # para mantener el estandar de VEE
                                # donde se evalua por un valor menor,
                                # podria cambiarse por singo (+)
                                # haciendo las modificaciones
                                # pertinentes.
    costo += interaccion
    return costo
#- Funcion objetivo o funcion de costo -#


########- Busqueda exhaustiva MAX-CUT (caso lineal) VEE -##########
def be_3_lineal_vee(corridas, costo_total, costo_actual,
                    costo_minimo, mejores_cuentas,
                    num_particulas, simulador_local):

    for i in np.arange(corridas): # Pruebas para valores de beta.

        # Valor de prueba para "beta" que corresponde al parametro
        # del operador mezclador.
        beta = 2*m.pi*(i/corridas)

        for j in np.arange(corridas): # Pruebas para valores de
                                      # gamma.

            # Valor de prueba para "gamma" que corresponde al
            # parametro del operador de fase.
            gamma = 2*m.pi*(j/corridas)

            # Variable para almacenar el "valor esperado".
            costo_aparicion = 0

            q_reg = QuantumRegister(num_particulas, name="q_reg")
            c_reg = ClassicalRegister(num_particulas, name="c_reg")
            q_c = QuantumCircuit(q_reg,c_reg,name="q_c")

            #* Paso (1)
            #* El primer paso es generar la superposicion de
            #* estados iniciales, esto se logra aplicando
            #* compuertas Hadamard a los qubits del registro
            #* cuantico.
            for q in np.arange(num_particulas):
                q_c.h(q_reg[int(q)]) # Aplicamos la compuerta "h"
                                    # (Hadamard) a los qubits.

            #* Paso (2)
            #* Se aplican las compuertas "Z" que corresponden
            #* al operador de fase, esto se realiza considerando
            #* las conexiones entre las particulas para el caso
            #* del Ising Spin Model y tambien se considera el
            #* valor de"gamma".
            for conexion in np.arange(num_particulas-1):
                q_c.cx(q_reg[conexion], q_reg[conexion+1])
                q_c.p(2*gamma, q_reg[conexion+1])
                q_c.cx(q_reg[conexion], q_reg[conexion+1])
```

```
80
81              #** Se omite el "Paso (3)" del Ising Spin Model
82              #** ya que no hay aporte de campo externo para
83              #** Max-Cut.
84
85              #* Paso (3)
86              #* Aqui se agrega el operador mezclador, que
87              #* corresponde a los procesos de interferencia
88              #* constructiva o destructiva tomando en cuenta
89              #* el valor de "beta".
90              for mez in np.arange(num_particulas):
91                  q_c.rx(beta, mez)
92
93              #* Paso (4)
94              #* Establecer las mediciones para el registro
95              #* cuantico, los valores de los estados base que
96              #* se obtienen al medir los qubits (q_reg) son
97              #* almacenados en los bits clasicos (c_reg).
98              q_c.barrier()
99              q_c.measure(q_reg, c_reg)
100
101             # Numero de experimentos que se correrar por cada
102             # par de valores gamma y beta.
103             n_experimentos = 1000
104
105             # Ejecutar el circuito cuantico en el simulador.
106             trabajo = execute(q_c, simulador_local,
107                               shots= n_experimentos)
108
109             # Extraer los resultados.
110             resultado = trabajo.result()
111
112             # Extraer los datos estadisticos.
113             cuentas = resultado.get_counts(q_c)
114
115             #* Aqui se usa la tecnica denominada
116             #* "VALOR DE ENERGIA ESPERADO (VEE)"
117             #* que se usa para conocer el
118             #* aporte total de todos los experimentos del
119             #* circuito para un numero determinado de
120             #* experimentos, usando la correcta combinacion
121             #* de parametros VEE debera tender al valor de la
122             #* funcion costo de evaluada con el estado optimo
123             #* del sistema.
124             for elemento in cuentas.items():
125                 costo = funcion_objetivo(elemento[0])
126                 costo_ap = costo*float(elemento[1])
127                 costo_aparicion += (costo_ap / n_experimentos)
128             if costo_aparicion <= costo_actual:
129                 costo_actual = costo_aparicion
130                 costo_minimo = [costo_actual, gamma, beta]
131                 mejores_cuentas = cuentas
132                 print("Costo actual [G1, B1]: ", costo_minimo)
133             else:
```

```
                continue

    return costo_minimo, mejores_cuentas
########- Busqueda exhaustiva MAX-CUT (caso lineal) VEE -##########


###-------------------------MAIN-------------------------------###
if __name__ == "__main__":
    #* Constantes de calibracion para QAOA *#
    corridas = 50
    costo_total = 0
    costo_actual = 0
    costo_minimo = [0.0, 0.0, 0.0]
    mejores_cuentas = []
    num_particulas = 3
    simulador_local = Aer.backends(name="qasm_simulator")[0]
    #* Funcion para calculo del circuito y aplicacion de BE.
    costo_minimo, mejores_cuentas = be_3_lineal_vee(corridas,
                                    costo_total, costo_actual,
                                    costo_minimo, mejores_cuentas,
                                    num_particulas, simulador_local
    )
    #* Imprimir y graficar el mejor resultado.
    print("Costo minimo [G1, B1]: ", costo_minimo)
    plot_histogram(mejores_cuentas)
    plt.show()
###-------------------------MAIN-------------------------------###
```

Listing B.7: Código para simulación local usando BE en el problema de Max-Cut de 3 nodos en configuración lineal.

El **Código B.7** muestra el programa usado para simular el primer problema de Max-Cut de 3 nodos en configuración lineal. Como puede observarse entre el *Paso (2)* y *Paso (3)* existe una pequeña nota donde se aclara cuál es la diferencia entre este programa con respecto del usado para ISM, la otra diferencia se presenta en la *Función objetivo* donde ya no existe aporte del campo magnético externo y el costo de evaluación de un estado solo toma en cuenta sus conexiones entre nodos. Los demás aspectos se comportan de igual forma que en el caso anterior de ISM para 3 partículas.

Existen también dos consideraciones que son importantes de mencionar, la primera relaciona al signo del resultado generado, ya que este resultado se establece con signo negativo porque la *Función objetivo* realiza las sumas de cada conexión con -1, sin embargo, el resultado final de la evaluación solo es necesario multiplicarlo por -1 para obtener su representación en los enteros positivos. La siguiente consideración es el signo que se maneja, por ejemplo para las **Ecuaciones 3.14** y **3.15** son con signo negativo, esto en realidad nos dice que existe una rotación de la fase global del sistema lo cual es solo una convención usada para abordar este tipo de problemas, de manera real no es posible distinguir este tipo

de rotaciones cuando se habla de operadores cuánticos y es por ello que se puede omitir el signo para utilizar un modelo similar al de ISM.

Código para problema de 4 nodos en configuración cíclica

Las modificaciones generadas al código base de 3 nodos se muestran a continuación, debe tomarse en cuenta que los valores para 4 nodos presentes en las *constantes de calibración* de QAOA se obvian, y debido a ello no son incluidas en el siguiente código.

```
################-- Problema de 4 nodos para Max-Cut --################
#####################################################################


############################### CAMBIOS ***############################

###############- Busqueda exhaustiva (caso ciclica) -################
    # .
    # .
    # .
            #* Paso (2)
            #* Se aplican las compuertas "Z" que corresponden
            #* al operador de fase, esto se realiza considerando
            #* las conexiones entre las particulas para el caso
            #* del Ising Spin Model y tambien se considera el
            #* valor de"gamma".
            for conexion in np.arange(num_particulas):
                q_c.cx(q_reg[conexion], q_reg[conexion-1])
                q_c.p(2*gamma, q_reg[conexion])
                q_c.cx(q_reg[conexion], q_reg[conexion-1])
            # .
            # .
            # .
###############- Busqueda exhaustiva (caso ciclica) -################

############################### CAMBIOS ***############################
```

Listing B.8: Código para simulación local usando BE en el problema de Max-Cut de 4 nodos en configuración cíclica.

Como se puede aprecia en el **Código B.8** la única variación que existe entre este programa y el anterior es en el *Paso (2)* en el cual se toman en cuenta la nueva conexión que existe entre el primer y último nodo.

Para los programas generados con relación al aumento a dos parámetros β_1 y β_2 del operador mezclador y el aumento a dos parejas de parámetros γ_1, γ_2, con β_1 y β_2 (dos operadores de fase y dos operadores mezcladores consecutivos) son muy similares a los usados para ISM (problemas de 4 partículas y 5 partículas), por ello no se considera relevante agregarlos explícitamente con la finalidad de mantener reducido el cuerpo de este apéndice.

Código para problema de 5 nodos en configuración completa

El último programa usando BE corresponde al código de 5 nodos en configuración completa de Max-Cut, el programa es muy similar a los anteriores donde el siguiente código muestra las modificaciones realizadas.

```
###############-- Problema de 5 nodos para Max-Cut --###############
#########################################################################


############################** CAMBIOS **############################

##############- Busqueda exhaustiva (caso completo) -################
    # .
    # .
    # .
            #* Paso (2)
            #* Se aplican las compuertas "Z" que corresponden
            #* al operador de fase, esto se realiza considerando
            #* las conexiones entre las particulas para el caso
            #* del Ising Spin Model y tambien se considera el
            #* valor de"gamma".
            conexiones = []
            for conexion in np.arange(num_particulas):
                for conexion_2 in np.arange(num_particulas):
                    if conexion != conexion_2:
                        if [conexion, conexion_2] and [conexion_2,
    conexion] not in conexiones:
                            q_c.cx(q_reg[conexion_2],
                                   q_reg[conexion])
                            q_c.p(2*gamma, q_reg[conexion_2])
                            q_c.cx(q_reg[conexion_2],
                                   q_reg[conexion])
                            conexiones.append([conexion,
                                               conexion_2])
                            conexiones.append([conexion_2,
                                               conexion])
                        else:
                            continue
                    else:
                        continue
            q_c.barrier()
            # .
            # .
            # .
##############- Busqueda exhaustiva (caso completo) -################

############################** CAMBIOS **############################
```

Listing B.9: Código para simulación local usando BE en el problema de Max-Cut de 5 nodos en configuración completa.

Como podemos ver nuevamente, la modificación que se genera solo es en el *Paso (2)* donde existe un aumento del número de conexiones entre nodos, para este caso es necesario generar conexiones entre todos los nodos del grafo y no crear repeticiones.

B.1.2. Programas usando método de Búsqueda Local Iterativa

El siguiente método de optimización implementado es BLI, este método es mucho más complejo que el método de BE. De manera general, esta heurística tiene cuatro etapas, la primera es el inicio aleatorio dentro del espacio de búsqueda, la segunda es la implementación de la búsqueda local dentro de un área determinada, la tercera es la aplicación recurrente de Ascenso en la Montaña Estocástico y por último está la cuarta etapa que se encarga de evitar la convergencia temprana de la BLI.

Programas para ISM

La primera clase de códigos descritos será para los problemas de ISM, donde se usará al primer problema (3 partículas en configuración lineal) como base y solo irán agregando las notas para las modificaciones realizadas para los otros casos.

Código para problema de 3 partículas en configuración lineal

El programa generado para el problema de 3 partículas se muestra a continuación.

```
#####-- Problema de 3 particulas para Ising Spin Model (BL) --#####
#########################################################################


#- Paqueteria complementaria para ejecucion de QAOA y -#
#- metodo de optimizacion.                            -#
from qiskit import QuantumRegister, ClassicalRegister
from qiskit import QuantumCircuit, Aer, execute
from qiskit.visualization import plot_histogram
import numpy as np
import matplotlib.pyplot as plt
import collections
import math as m
import random


#- Funcion de verificacion -#
def dentro_rango(punto, limites):
    # Checar si un valor propuesto esta dentro del rango
```

```
    # permitido.
    if punto[0] >= limites[0] and punto[0] <= limites[1]:
        if punto[1] >= limites[0] and punto[1] <= limites[1]:
            return True
        else:
            return False
    else:
        return False
#- Funcion de verificacion -#


#- Algoritmo de Ascenso en la Montana Estocastico -#
def ame(limites, n_iteraciones, tam_paso, punto_inicial,
        costo_total, costo_actual, costo_minimo,
        mejores_cuentas, h, num_particulas, simulador_local):

    mejores_cuentas = []
    # Se guarda el punto inicial.
    solucion = punto_inicial
    # Evaluar el punto inicial.
    eval_solucion, cuentas = bl_3_lineal(solucion, costo_total,
                            costo_actual, costo_minimo,
                            mejores_cuentas, h, num_particulas,
                            simulador_local)
    # Aplicar el Ascenso en la Montana.
    for i in range(n_iteraciones):
        # Usar las interacciones.
        candidato = None
        while candidato is None or not dentro_rango(candidato,
                                                    limites):
            eleccion_a = np.random.choice([0,1])
            if eleccion_a == 0:
                candidato_1 = solucion[0] +\
                              np.random.uniform(limites[0],
                                                limites[1]) *\
                              np.random.uniform(tam_paso[0],
                                                tam_paso[1])
                candidato_2 = solucion[1]
            else:
                candidato_1 = solucion[0]
                candidato_2 = solucion[1] +\
                              np.random.uniform(limites[0],
                                                limites[1]) *\
                              np.random.uniform(tam_paso[0],
                                                tam_paso[1])

            candidato = [candidato_1, candidato_2]
        # Evaluar el punto canditato
        eval_candidato, cuentas = bl_3_lineal(candidato,costo_total
    ,
                                              costo_actual,
                                              costo_minimo,
                                              mejores_cuentas, h,
                                              num_particulas,
```

```
                                                simulador_local)
        # Checar si el candidato es mejor que la solucion previa.
        if eval_candidato[0] < eval_solucion[0]:
            # Guardar la solucion y la evaluacion.
            solucion = candidato
            eval_solucion = eval_candidato
            mejores_cuentas = cuentas

    return solucion, eval_solucion, mejores_cuentas
#- Algoritmo de Ascenso en la Montana Estocastico -#


#- Algoritmo de Busqueda Local Iterativa -#
def busqueda_local_iterada_3p(disp_conv, valor_conv,
                          limites, n_iteraciones,
                          tam_paso, n_reinicios, tam_per,
                          costo_total, costo_actual,
                          costo_minimo, mejores_cuentas, h,
                          num_particulas, simulador_local):
    # Variables para el reinicio por convergencia.
    contador_conv = 0
    eval_act_consec = 0 # Guarda el valor actual que se esta
                        # evaluando para establecer si es de
                        # forma consecutiva.
    dist_vals = []
    # Definir el punto de inicio.
    mejor = None
    while mejor is None or not dentro_rango(mejor, limites):
        mejor_1 = np.random.uniform(limites[0], limites[1])
        mejor_2 = np.random.uniform(limites[0], limites[1])
        mejor = [mejor_1, mejor_2]
    # Calcular los valores para el "mejor punto actual".
    mejor_eval, cuentas = bl_3_lineal(mejor, costo_total,
                                      costo_actual,
                costo_minimo, mejores_cuentas, h,
                num_particulas, simulador_local)
    # El valor consecutivo de convergencia toma primero el
    # valor de "mejor_eval[0]".
    eval_act_consec = mejor_eval[0]
    # Se repite el proceso el numero de veces que se especifique.
    for n in range(n_reinicios):
        # Generando un nuevo punto de inicio, usando el mejor valor
        # actual y la perturbacion.
        punto_inicio = None
        while punto_inicio is None or not dentro_rango(punto_inicio
    ,
                                                       limites):
            eleccion_b = np.random.choice([0,1])
            if eleccion_b == 0:
                punto_inicio_1 = mejor[0] +\
                         np.random.uniform(limites[0], limites[1])
    *\
                         np.random.uniform(tam_per[0], tam_per[1])
                punto_inicio_2 = mejor[1]
```

```
125             else:
126                 punto_inicio_1 = mejor[0]
127                 punto_inicio_2 = mejor[1] +\
128                             np.random.uniform(limites[0],
129                                               limites[1]) *\
130                             np.random.uniform(tam_per[0],
131                                               tam_per[1])
132 
133             punto_inicio = [punto_inicio_1, punto_inicio_2]
134         # Realizar el Ascenso en la Montana Estocastico (ame).
135         solucion, eval_solucion, cuentas = ame(limites,
    n_iteraciones,
136                                                tam_paso,
    punto_inicio,
137                                                costo_total,
138                                                costo_actual,
139                                                costo_minimo,
140                                                mejores_cuentas, h,
141                                        num_particulas,
    simulador_local)
142 ########################################################################
143         #-------------- REINICIOS POR CONVERGENCIA ---------------#
144         # Iniciar un contador para reiniciar el punto de
145         # exploracion despues de un cierto numero de repeticiones
146         # de un criterio, esto se hace para evitar que el algoritmo
147         # converga muy pronto.
148 
149         # Se mide la "distancia" que existe entre la evaluacion
150         # pasada y la actual.
151 
152         delta_eval = eval_solucion[0] - eval_act_consec
153         abs_delta_eval = abs(delta_eval)
154         dist_vals.append(abs_delta_eval)
155         contador_conv += 1
156         contador_salto = 0
157         if contador_conv == disp_conv: # Si se cumple el criterio
158                                        # del numero mediciones, se
159                                        # calcula el promedio.
160             radio_exploracion = np.sum(dist_vals)/len(dist_vals)
161             print("Radio exploracion: ", radio_exploracion)
162 
163             # Si el radio de exploracion es menor o igual al valor
164             # gatillo se genera el protocolo para reiniciar la
165             # busqueda de manera aleatoria.
166             if radio_exploracion <= valor_conv:
167                 # Definir el punto de inicio.
168                 mejor_n = None
169                 eval_solucion_costo = 99999
170                 eval_final = 0
171                 while mejor_n is None or not dentro_rango(mejor_n,
172                                                           limites):
173                     mejor_1 = np.random.uniform(limites[0],
174                                                 limites[1])
175                     mejor_2 = np.random.uniform(limites[0],
```

```
                                                limites[1])
                    mejor_n = [mejor_1, mejor_2]
                    if dentro_rango(mejor_n, limites) == True:
                        # Se realiza la evaluacion de ese punto
                        # aleatorio.
                        eval_solucion_n, cuentas_n = bl_3_lineal(
    mejor_n,
                                                costo_total, costo_actual,
                                                costo_minimo,
    mejores_cuentas, h,
                                                num_particulas,
    simulador_local)
                        eval_solucion_costo = eval_solucion_n[0]

                        # Guardar la mejor solucion explorada hasta
                        # el momento.
                        if eval_solucion_n[0] < eval_final:
                            mejor_n = eval_solucion_n[1:]
                            mejor = mejor_n
                            eval_solucion = eval_solucion_n
                            cuentas = cuentas_n
                            eval_final = eval_sulucion_n[0]

                        if eval_solucion_costo < mejor_eval[0]:
                            print("Salto: ", eval_solucion_n)
                            # El salto genera los nuevos valores
                            # para iniciar la siguiente iteracion.
                            mejor_n = eval_solucion_n[1:]
                            print("Mejor_n: ", mejor_n)
                            # En este caso se encontro una
                            # combinacion "mejor" que tiene un
                            # resultado mejor de VEE, entonces los
                            # valores de los parametros (mejor),
                            # la evaluacion de esta combinacion
                            # (mejor_eval) y las cuentas
                            # (mejores_cuentas) pasan
                            # a ser el mejor resultado actual.
                            mejor = mejor_n
                            eval_solucion = eval_solucion_n
                            cuentas = cuentas_n
                            break
                        else:
                            # Permite mantener la exploracion,
                            # generando valores por fuera de los
                            # limites que mantienen trabajando al
                            # "while loop".
                            mejor_1 = limites[1]+1
                            mejor_2 = limites[1]+1
                            mejor_n = [mejor_1, mejor_2]

                        if contador_salto == 1000:
                            print("No reinicio mejor,reinicio
    aleatorio")
                            mejor_n = eval_solucion_n[1:]
```

```
                                print("Valores: ", mejor_n)
                                # Se toma el inicio "mejor" como la
                                # combinacion nueva de parametros
                                # segun el salto, pero se considera
                                # como solo una solucion que sera
                                # comparada, en caso de no se mejor
                                # solo se tomara el punto "mejor"
                                # como nuestro nuevo inicio de
                                # exploracion pero aun almacenando
                                # la mejor solucion global
                                mejor = mejor_n
                                eval_solucion = eval_solucion_n
                                cuentas = cuentas_n
                                solucion = mejor_n
                                contador_salto = 0
                                eval_final = 0
                                break

                            contador_salto += 1 # Contador en caso de
                                                # no encontrar mejor
                                                # salto.
                        else:
                            continue
                    # Reiniciar contador de convergencia.
                    contador_conv = 0
                    dist_vals = []
                else:
                    contador_conv = 0
                    dist_vals = []
            #-------------------------------------------------------------#
            # Evaluar para el "nuevo mejor" si existe.
            if eval_solucion[0] < mejor_eval[0]:
                mejor, mejor_eval = solucion, eval_solucion
                mejores_cuentas = cuentas
                print("Reinicio--> N: ", n, ", M_Ev: ", mejor_eval)
                print("Cuentas: ", cuentas)
                print("\n")
                eval_act_consec = mejor_eval[0]
    return mejor, mejor_eval, mejores_cuentas
#- Algoritmo de Busqueda Local Iterativa -#


#- Funcion objetivo o funcion de costo -#
def funcion_objetivo(x, h):
    costo = 0
    interaccion = 0
    campo_mag = 0
    # Calculo del costo de interaccion.
    for i in range(len(x)-1):
        if x[i] == x[i+1]:
            interaccion -= 1
        else:
            interaccion += 1
    costo += interaccion
```

```
280     # Calculo del costo de campo magnetico externo.
281     for i in range(len(x)):
282         if h[i] > 0:
283             if int(x[i]) == 0:
284                 campo_mag = campo_mag + (-1*h[i])
285             else:
286                 campo_mag = campo_mag + (1*h[i])
287         if h[i] < 0:
288             if int(x[i]) == 0:
289                 campo_mag = campo_mag + (-1*h[i])
290             else:
291                 campo_mag = campo_mag + (1*h[i])
292     costo += campo_mag
293     return costo
294 #- Funcion objetivo o funcion de costo -#
295
296 ##################- Busqueda local (caso lineal) -##################
297 def bl_3_lineal(candidato, costo_total, costo_actual,
298                 costo_minimo, mejores_cuentas, h,
299                 num_particulas, simulador_local):
300
301     # Valor de prueba para "beta" que corresponde al parametro
302     # del primer operador mezclador.
303     beta = candidato[1]
304
305     # Valor de prueba para "gamma" que corresponde al parametro
306     # del primer operador de fase.
307     gamma = candidato[0]
308
309     # Valor para almacenar el valor de energia esperado para
310     # una iteracion.
311     costo_aparicion = 0
312
313     # Generacion de regirsto cuantico "q_reg" y registro
314     # clasico "c_reg".
315     q_reg = QuantumRegister(num_particulas, name="q_reg")
316     c_reg = ClassicalRegister(num_particulas, name="c_reg")
317     q_c = QuantumCircuit(q_reg,c_reg,name="q_c")
318
319     #* Paso (1)
320     #* El primer paso es generar la superposicion de estados
321     #* iniciales, esto se logra aplicando compuertas Hadamard
322     #* a los qubits del registro cuantico.
323     for q in np.arange(num_particulas):
324         q_c.h(q_reg[int(q)]) # Aplicamos la compuerta "h"
325                             # (Hadamard) a los qubits.
326
327     #* Paso (2)
328     #* Se aplican las compuertas "Z" que corresponden al
329     #* operador de fase, esto se realiza considerando las
330     #* conexiones entre las particulas para el caso del
331     #* Ising Spin Model y tambien se considera el valor de
332     #* "gamma".
333     for conexion in np.arange(num_particulas-1):
```

```
334         q_c.cx(q_reg[conexion], q_reg[conexion+1])
335         q_c.p(2*gamma, q_reg[conexion+1])
336         q_c.cx(q_reg[conexion], q_reg[conexion+1])
337 
338     #* Paso (3)
339     #* Despues de realizar las conexiones entre particulas,
340     #* se procede a agregar el campo magnetico externo "h"
341     #* para cada padricula, esto corresponde al segundo
342     #* termino del operador de fase.
343     for campo in np.arange(num_particulas):
344         q_c.p(gamma*h[int(campo)], campo)
345 
346     #* Paso (4)
347     #* Aqui se agrega el operador mezclador, que corresponde
348     #* a los procesos de interferencia constructiva o
349     #* destructiva tomando en cuenta el valor de "beta".
350     for mez in np.arange(num_particulas):
351         q_c.rx(beta, mez)
352 
353     #* Paso (5)
354     #* Establecer las mediciones para el registro cuantico,
355     #* los valores de los estados base que se obtienen al
356     #* medir los qubits (registro cuantico) son almacenados
357     #* en los bits clasicos (registro clasico).
358     q_c.barrier()
359     q_c.measure(q_reg, c_reg)
360 
361     # Numero de experimentos que se correrar por cada par
362     # de valores gamma y beta.
363     n_experimentos = 1000
364 
365     # Ejecutar el circuito cuantico en el simulador.
366     trabajo = execute(q_c, simulador_local,
367                       shots= n_experimentos)
368 
369     # Extraer los resultados.
370     resultado = trabajo.result()
371 
372     # Extraer los datos estadisticos.
373     cuentas = resultado.get_counts(q_c)
374 
375     for elemento in cuentas.items():
376         costo = funcion_objetivo(elemento[0], h)
377         costo_ap = costo*float(elemento[1])
378         costo_aparicion += (costo_ap / n_experimentos)
379 
380     costo_actual = costo_aparicion
381     costo_minimo = [costo_actual, gamma, beta]
382     mejores_cuentas = cuentas
383 
384     return costo_minimo, mejores_cuentas
385 ##################- Busqueda local (caso lineal) -###################
386 
387 
```

```
388 ###--------------------------- MAIN -----------------------------###
389 if __name__ == "__main__":
390
391     #- Parametros para la busqueda local -#
392     # Limites de valores para "gamma" y "beta".
393     limites = [0, 2*m.pi]
394     # Total de iteraciones (para "ame").
395     n_iteraciones = 1000
396     # Tamano maximo de paso.
397     tam_paso = [-0.05, 0.05]
398     # Numero de reinicios.
399     n_reinicios = 100
400     # Tamano de perturbacion.
401     tam_per = [0.0, 1.0]
402     # Valores para el reinicio por convergencia.
403     valor_conv = 1.0 # Diferencia entre la respuesta evaluada y la
404                      # mejor respuesta.
405     disp_conv = 10 # Numero de ocurrencias de "valor_conv" para que
406                    # se pueda disparar el reinicio aleatorio.
407     #-------------------------------------#
408
409     #- Parametros para circuito cuantico -#
410     costo_total = 0
411     costo_actual = 0
412     costo_minimo = [0.0, 0.0, 0.0]
413     mejores_cuentas = []
414     h = [0.7, 1.2, -1.6] # Campos magneticos externos
415                          # por particula.
416     num_particulas = 3
417     simulador_local = Aer.backends(name="qasm_simulator")[0]
418     #-------------------------------------#
419
420     mejor, mejor_eval, mejores_cuentas = busqueda_local_iterada_3p(
        disp_conv,
421                     valor_conv, limites, n_iteraciones, tam_paso,
422                     n_reinicios, tam_per, costo_total, costo_actual
        ,
423                     costo_minimo, mejores_cuentas, h,
        num_particulas,
424                     simulador_local)
425
426     print("Mejor (G1, B1): ", mejor)
427     print("Mejor evaluacion: ", mejor_eval)
428     plot_histogram(mejores_cuentas)
429     plt.show()
430 ###--------------------------- MAIN -----------------------------###
```

Listing B.10: Código para simulación local usando BLI en el problema de ISM de 3 partículas en configuración lineal.

El **Código B.10** describe todas las etapas necesarias para implementar el método de Búsqueda Local Iterativa como optimización para los distintos parámetros de los operadores de fase y mezclador, el programa contiene todas la

funciones necesarias en el mismo bloque, esto puede realizarse de distintas formas, ya que pueden implementarse paqueterías con las funciones de BLI y solo agregar el programa de QAOA, o incluso realizarlo usando programación orientada a objetos; pero debido a que normalmente la programación de algoritmos cuánticos se realiza usando programación procedural o procedimental, se aconseja que si las implementaciones son simples es recomendable hacerlo de forma procedural.

Pasando a la descripción del código tenemos las funciones complementarias iniciando con *Función de verificación* que se encarga de evaluar si un punto dado (en este caso de dos variables que corresponden a los parámetros γ y β) está dentro de los límites que corresponde al rango 0 a 2π. La siguiente función corresponde al *Algoritmo de Ascenso en la Montaña Estocástico* que sirve para realizar las búsquedas en áreas específicas a partir del punto generado por la BLI. Después se tiene la función de *Algoritmo de Búsqueda Local Iterativa*, esta sección del código se encarga de realizar la búsqueda local, que podría interpretarse como una búsqueda aleatoria en un área del espacio de búsqueda, esta área puede ir moviéndose o saltar en espacios cercanos al área de búsqueda inicial, internamente existe una etapa del algoritmo llamada *Reinicios por convergencia*, este paso no existe como tal en el algoritmo clásico de BLI, sin embargo, para garantizar que la búsqueda no converja antes de tiempo. Normalmente, los espacios vectoriales que contienen a los estados generados por los algoritmos cuánticos son del tipo de espacio de Hilbert, entonces los parámetros de γ y β generados por BLI mapean al espacio de Hilbert para generar en consecuencia un estado cuántico, por esto el espacio de búsqueda debe tener una estructura bastante compleja que podría catalogarse de forma general como no-convexa en la mayoría de los casos. Posteriormente, están las dos funciones que corresponden a la generación de QAOA, se tiene primero la función *Función de costo* que evalúa el costo en lo que se refiere al estado medido y la segunda función llamada *bl_3_lineal* que es el algoritmo para generar el circuito cuántico de QAOA, que como puede observarse no contiene a los ciclos **For loop** de Búsqueda Exhaustiva, ahora solo se tiene en las líneas 303 y 307 los "candidatos" que son los valores del punto generado por BLI que serán probados para QAOA y obtener un resultado estadístico de 1000 experimentos para ese punto (de dos valores) en específico. Finalmente, se tiene la función *MAIN* que contiene los valores para la calibración de BLI y QAOA, además, aquí se llama a la función *busqueda_local_iterada_3p* que realiza todo el proceso de BLI y algoritmo QAOA.

Código para problema de 4 partículas en configuración cíclica

Para el código del problema de 4 partículas en configuración cíclica usando BLI, solo se tienen unas pequeñas modificaciones al igual que en el caso de BE.

```
1 ########-- Problema de 4 particulas para Ising Spin Model --#######
2 ########--              usando Busqueda Local              --#######
```

```
#####################################################################


############################*** CAMBIOS ***############################


#- Algoritmo de Busqueda Local Iterativa -#
def busqueda_local_iterada_4p(limites, n_iteraciones,
                          tam_paso, n_reinicios, tam_per,
                          costo_total, costo_actual,
                          costo_minimo, mejores_cuentas, h,
                          num_particulas, simulador_local):

#- Funcion objetivo o funcion de costo -#
def funcion_objetivo(x, h):
    # .
    # .
    # .
    # Calculo del costo de interaccion.
    for i in range(len(x)):
        if x[i] == x[i-1]:
            interaccion -= 1
        else:
            interaccion += 1
    costo += interaccion
    # .
    # .
    # .
##################- Busqueda local (caso ciclico) -##################
    # .
    # .
    # .
    #* Paso (2)
    #* Se aplican las compuertas "Z" que corresponden al
    #* operador de fase, esto se realiza considerando las
    #* conexiones entre las particulas para el caso del
    #* Ising Spin Model y tambien se considera el valor de
    #* "gamma".
    for conexion in np.arange(num_particulas):
        q_c.cx(q_reg[conexion], q_reg[conexion-1])
        q_c.p(2*gamma, q_reg[conexion])
        q_c.cx(q_reg[conexion], q_reg[conexion-1])
    # .
    # .
    # .
##################- Busqueda local (caso ciclico) -##################


############################*** CAMBIOS ***############################
```

Listing B.11: Código para simulación local usando BLI en el problema de ISM de 4 partículas en configuración cíclica.

Como se ve en el código anterior, de igual forma que en el caso de BE solo

se modifica la función de evaluación de costo y el operador de fase del circuito cuántico para agregar la conexión extra de partículas.

Para el caso de 2 parámetros β en el operador mezclador, se tienen las modificaciones:

```
########-- Problema de 4 particulas para Ising Spin Model --#######
########--              usando Busqueda Local              --#######
########--                  con dos BETAs                  --#######
#########################################################################


############################*** CAMBIOS ***############################

#- Funcion de verificacion -#
def dentro_rango(punto, limites):
    # Checar si un valor propuesto esta dentro del rango
    # permitido.
    if punto[0] >= limites[0] and punto[0] <= limites[1]:
        if punto[1] >= limites[0] and punto[1] <= limites[1]:
            if punto[2] >= limites[0] and punto[2] <= limites[1]:
                return True
            else:
                return False
        else:
            return False
    else:
        return False
#- Funcion de verificacion -#


#- Algoritmo de Ascenso en la Montana Estocastico -#
        # .
        # .
        # .
        while candidato is None or not dentro_rango(candidato,
                                                    limites):
            eleccion_a = np.random.choice([0,1,2])
            if eleccion_a == 0:
                candidato_1 = solucion[0] +\
                              np.random.uniform(limites[0],
                                                limites[1]) *\
                              np.random.uniform(tam_paso[0],
                                                tam_paso[1])
                candidato_2 = solucion[1]
                candidato_3 = solucion[2]
            elif eleccion_a == 1:
                candidato_1 = solucion[0]
                candidato_2 = solucion[1] +\
                              np.random.uniform(limites[0],
                                                limites[1]) *\
                              np.random.uniform(tam_paso[0],
                                                tam_paso[1])
```

```
48                  candidato_3 = solucion[2]
49              else:
50                  candidato_1 = solucion[0]
51                  candidato_2 = solucion[1]
52                  candidato_3 = solucion[2] +\
53                                np.random.uniform(limites[0],
54                                                  limites[1]) *\
55                                np.random.uniform(tam_paso[0],
56                                                  tam_paso[1])
57
58              candidato = [candidato_1, candidato_2, candidato_3]
59              # .
60              # .
61              # .
62  #- Algoritmo de Ascenso en la Montana Estocastico -#
63
64  #- Algoritmo de Busqueda Local Iterativa -#
65  def busqueda_local_iterada_4p(limites, n_iteraciones,
66                            tam_paso, n_reinicios, tam_per,
67                            costo_total, costo_actual,
68                            costo_minimo, mejores_cuentas, h,
69                            num_particulas, simulador_local):
70      # .
71      # .
72      # .
73      mejor = None
74      while mejor is None or not dentro_rango(mejor, limites):
75          mejor_1 = np.random.uniform(limites[0], limites[1])
76          mejor_2 = np.random.uniform(limites[0], limites[1])
77          mejor_3 = np.random.uniform(limites[0], limites[1])
78          mejor = [mejor_1, mejor_2, mejor_3]
79      # .
80      # .
81      # .
82          punto_inicio = None
83          while punto_inicio is None or not dentro_rango(punto_inicio
        , limites):
84              eleccion_b = np.random.choice([0,1,2])
85              if eleccion_b == 0:
86                  punto_inicio_1 = mejor[0] +\
87                           np.random.uniform(limites[0],
88                                             limites[1]) *\
89                           np.random.uniform(tam_per[0],
90                                             tam_per[1])
91                  punto_inicio_2 = mejor[1]
92                  punto_inicio_3 = mejor[2]
93              elif eleccion_b == 1:
94                  punto_inicio_1 = mejor[0]
95                  punto_inicio_2 = mejor[1] +\
96                               np.random.uniform(limites[0],
97                                                 limites[1]) *\
98                               np.random.uniform(tam_per[0],
99                                                 tam_per[1])
100                 punto_inicio_3 = mejor[2]
```

```
101             else:
102                 punto_inicio_1 = mejor[0]
103                 punto_inicio_2 = mejor[1]
104                 punto_inicio_3 = mejor[2] +\
105                             np.random.uniform(limites[0],
106                                               limites[1]) *\
107                             np.random.uniform(tam_per[0],
108                                               tam_per[1])
109             punto_inicio = [punto_inicio_1, punto_inicio_2,
110                             punto_inicio_3]
111             # .
112             # .
113             # .
114             #----------- REINICIOS POR CONVERGENCIA --------------#
115             # .
116             # .
117             # .
118                 while mejor_n is None or not dentro_rango(mejor_n,
119                                                           limites):
120                     mejor_1 = np.random.uniform(limites[0],
121                                                 limites[1])
122                     mejor_2 = np.random.uniform(limites[0],
123                                                 limites[1])
124                     mejor_3 = np.random.uniform(limites[0],
125                                                 limites[1])
126                     mejor_n = [mejor_1, mejor_2, mejor_3]
127                     # .
128                     # .
129                     # .
130                         else:
131                             # Permite mantener la exploracion,
132                             # generando valores por fuera de los
133                             # limites que mantienen trabajando al
134                             # "while loop".
135                             mejor_1 = limites[1]+1
136                             mejor_2 = limites[1]+1
137                             mejor_3 = limites[1]+1
138                             mejor_n = [mejor_1, mejor_2, mejor_3]
139                             # .
140                             # .
141                             # .
142             #----------- REINICIOS POR CONVERGENCIA --------------#
143
144
145 ##################- Busqueda local (caso ciclico) -##################
146     # .
147     # .
148     # .
149     #********************* Paso (4.5) OPCIONAL *********************#
150     #* Este paso se realiza para aumentar el numero de parametros
151     #* dentro del operador mezclador, es importante agregar el
152     #* nuevo parametro (ej. "beta_2") de manera entrelazada para
153     #* garantizar que QAOA sea capaz de generar estados
154     #* entrelazados y que esto a su vez aumente el espacio de
```

```
155     #* busqueda posible del algoritmo.
156     for ent in np.arange(num_particulas):
157         q_c.cx(q_reg[ent], q_reg[ent-1])
158     q_c.barrier()
159     #* Repetir Paso (4)
160     for mez2 in np.arange(num_particulas):
161         q_c.rx(beta_2, mez2)
162     q_c.barrier()
163     #*************************************************************#
164     # .
165     # .
166     # .
167 ##################- Busqueda local (caso ciclico) -#################
168 
169 ############################*** CAMBIOS ***############################
```

Listing B.12: Código para simulación local usando BLI en el problema de ISM de 4 partículas en configuración cíclica.

En el **Código B.12** se muestra que el aumento de un parámetro usando BLI tienen varias modificaciones a lo largo de todo el programa, esto se da porque se tiene que contemplar la aplicación del parámetro en BLI y en AME, así mismo se aplican las modificaciones pertinentes para el QAOA y la función de evaluación de costo al igual que en BE.

Código para problema de 5 partículas en configuración completa

Las modificaciones realizadas para el problema de 5 partículas en configuración completa son las siguientes.

```
1  ########-- Problema de 5 particulas para Ising Spin Model --#######
2  ########--              usando Busqueda Local              --#######
3  ######################################################################
4  
5  
6  ############################*** CAMBIOS ***############################
7  
8  #- Algoritmo de Busqueda Local Iterativa -#
9  def busqueda_local_iterada_5p(disp_conv, valor_conv,
10                            limites, n_iteraciones,
11                            tam_paso, n_reinicios, tam_per,
12                            costo_total, costo_actual,
13                            costo_minimo, mejores_cuentas, h,
14                            num_particulas, simulador_local):
15 
16 #- Funcion objetivo o funcion de costo -#
17 def funcion_objetivo(x, h):
18     # .
19     # .
20     # .
```

```
    # Calculo del costo de interaccion.
    interacciones = []
    for i in range(len(x)):
        for j in range(len(x)):
            if i != j:
                if [i,j] and [j,i] not in interacciones:
                    if x[i] == x[j]:
                        interaccion -= 1
                        interacciones.append([i,j])
                        interacciones.append([j,i])
                    else:
                        interaccion += 1
                        interacciones.append([i,j])
                        interacciones.append([j,i])
                else:
                    continue
            else:
                continue
    costo += interaccion
    # .
    # .
    # .
#################- Busqueda local (caso completo) -#################
    # .
    # .
    # .
    #* Paso (2)
    #* Se aplican las compuertas "Z" que corresponden al
    #* operador de fase, esto se realiza considerando las
    #* conexiones entre las particulas para el caso del
    #* Ising Spin Model y tambien se considera el valor de
    #* "gamma".
    conexiones = []
    for conexion in np.arange(num_particulas):
        for conexion_2 in np.arange(num_particulas):
            if conexion != conexion_2:
                if [conexion, conexion_2] and [conexion_2, conexion
    ] not in conexiones:
                    q_c.cx(q_reg[conexion_2], q_reg[conexion])
                    q_c.p(2*gamma, q_reg[conexion_2])
                    q_c.cx(q_reg[conexion_2], q_reg[conexion])
                    conexiones.append([conexion, conexion_2])
                    conexiones.append([conexion_2, conexion])
                else:
                    continue
            else:
                continue
    q_c.barrier()
    # .
    # .
    # .
################- Busqueda local (caso completo) -##################

```

```
73 ############################*** CAMBIOS ***############################
```

Listing B.13: Código para simulación local usando BLI en el problema de ISM de 5 partículas en configuración completa.

Al igual que para el caso de 4 partículas se tienen modificaciones para la función de evaluación y para el operador de fase de QAOA.

Para los casos de modificación de QAOA con 2 parámetros en el operador mezclador y modificación con dos operadores de fase y dos operadores mezcladores (uso de 4 parámetros), usan la misma estrategia que para el caso de 4 partículas con dos parámetros del operador mezclador, es decir, que se incorporan esas nuevas variables para explorar durante las distintas etapas de la BLI, AME y las funciones propias de QAOA.

Programas para Max-Cut

Por último, los programas usados para Max-Cut incorporando BLI se muestran a continuación, cabe la pena señalar que la mayoría de ellos son solo modificaciones de ciertos aspectos del circuito de QAOA y la función de evaluación, ya que la estrategia de BLI se puede aplicar de igual forma independientemente del problema que se explore, donde las únicas modificaciones en BLI se dan en el número de variables exploradas.

Código para problema de 3 nodos en configuración lineal

El primer código muestra el problema de 3 nodos en configuración lineal usando BLI, este programa se usará como base para solo aclarar las modificaciones para los siguientes casos.

```
1  #####--------- Problema de 3 nodos para Max-Cut (BL) ---------#####
2  ######################################################################
3
4
5  #- Paqueteria complementaria para ejecucion de QAOA y -#
6  #- metodo de optimizacion.                            -#
7  from qiskit import QuantumRegister, ClassicalRegister
8  from qiskit import QuantumCircuit, Aer, execute
9  from qiskit.visualization import plot_histogram
10 import numpy as np
11 import matplotlib.pyplot as plt
12 import collections
13 import math as m
14 import random
15
16
17 #- Funcion de verificacion -#
```

```
18 def dentro_rango(punto, limites):
19     # Checar si un valor propuesto esta dentro del rango
20     # permitido.
21     if punto[0] >= limites[0] and punto[0] <= limites[1]:
22         if punto[1] >= limites[0] and punto[1] <= limites[1]:
23             return True
24         else:
25             return False
26     else:
27         return False
28 #- Funcion de verificacion -#
29
30
31 #- Algoritmo de Ascenso en la Montana Estocastico -#
32 def ame(limites, n_iteraciones, tam_paso, punto_inicial,
33         costo_total, costo_actual, costo_minimo,
34         mejores_cuentas, num_particulas, simulador_local):
35
36     mejores_cuentas = []
37     # Se guarda el punto inicial.
38     solucion = punto_inicial
39     # Evaluar el punto inicial.
40     eval_solucion, cuentas = bl_3_lineal(solucion, costo_total,
41                              costo_actual, costo_minimo,
42                              mejores_cuentas, num_particulas,
43                              simulador_local)
44     # Aplicar el Ascenso en la Montana.
45     for i in range(n_iteraciones):
46         # Usar las interacciones.
47         candidato = None
48         while candidato is None or not dentro_rango(candidato,
49                                                     limites):
50             eleccion_a = np.random.choice([0,1])
51             if eleccion_a == 0:
52                 candidato_1 = solucion[0] +\
53                               np.random.uniform(limites[0],
54                                                 limites[1]) *\
55                               np.random.uniform(tam_paso[0],
56                                                 tam_paso[1])
57                 candidato_2 = solucion[1]
58             else:
59                 candidato_1 = solucion[0]
60                 candidato_2 = solucion[1] +\
61                               np.random.uniform(limites[0],
62                                                 limites[1]) *\
63                               np.random.uniform(tam_paso[0],
64                                                 tam_paso[1])
65
66             candidato = [candidato_1, candidato_2]
67         # Evaluar el punto canditato
68         eval_candidato, cuentas = bl_3_lineal(candidato,costo_total
    ,
69                                               costo_actual,
70                                               costo_minimo,
```

```
                                                mejores_cuentas,
                                                num_particulas,
                                                simulador_local)
        # Checar si el candidato es mejor que la solucion previa.
        if eval_candidato[0] < eval_solucion[0]:
            # Guardar la solucion y la evaluacion.
            solucion = candidato
            eval_solucion = eval_candidato
            mejores_cuentas = cuentas

    return solucion, eval_solucion, mejores_cuentas
#- Algoritmo de Ascenso en la Montana Estocastico -#


#- Algoritmo de Busqueda Local Iterativa -#
def busqueda_local_iterada_3p(disp_conv, valor_conv,
                          limites, n_iteraciones,
                          tam_paso, n_reinicios, tam_per,
                          costo_total, costo_actual,
                          costo_minimo, mejores_cuentas,
                          num_particulas, simulador_local):
    # Variables para el reinicio por convergencia.
    contador_conv = 0
    eval_act_consec = 0 # Guarda el valor actual que se esta
                        # evaluando para establecer si es de
                        # forma consecutiva.
    dist_vals = []
    # Definir el punto de inicio.
    mejor = None
    while mejor is None or not dentro_rango(mejor, limites):
        mejor_1 = np.random.uniform(limites[0], limites[1])
        mejor_2 = np.random.uniform(limites[0], limites[1])
        mejor = [mejor_1, mejor_2]
    # Calcular los valores para el "mejor punto actual".
    mejor_eval, cuentas = bl_3_lineal(mejor, costo_total,
                                      costo_actual,
                                      costo_minimo,
                                      mejores_cuentas,
                                      num_particulas,
                                      simulador_local)
    # El valor consecutivo de convergencia toma primero el
    # valor de "mejor_eval[0]".
    eval_act_consec = mejor_eval[0]
    # Se repite el proceso el numero de veces que se especifique.
    for n in range(n_reinicios):
        # Generando un nuevo punto de inicio, usando el mejor valor
        # actual y la perturbacion.
        punto_inicio = None
        while punto_inicio is None or not dentro_rango(punto_inicio
    ,
                                                       limites):
            eleccion_b = np.random.choice([0,1])
            if eleccion_b == 0:
                punto_inicio_1 = mejor[0] +\
```

```
                        np.random.uniform(limites[0], limites[1])
    *\
                        np.random.uniform(tam_per[0], tam_per[1])
                punto_inicio_2 = mejor[1]
            else:
                punto_inicio_1 = mejor[0]
                punto_inicio_2 = mejor[1] +\
                            np.random.uniform(limites[0],
                                              limites[1]) *\
                            np.random.uniform(tam_per[0],
                                              tam_per[1])

            punto_inicio = [punto_inicio_1, punto_inicio_2]
        # Realizar el Ascenso en la Montana Estocastico (ame).
        solucion, eval_solucion, cuentas = ame(limites,
                                               n_iteraciones,
                                               tam_paso,
                                               punto_inicio,
                                               costo_total,
                                               costo_actual,
                                               costo_minimo,
                                               mejores_cuentas,
                                               num_particulas,
                                               simulador_local)
        #-------------- REINICIOS POR CONVERGENCIA ---------------#
        # Iniciar un contador para reiniciar el punto de
        # exploracion despues de un cierto numero de repeticiones
        # de un criterio, esto se hace para evitar que el algoritmo
        # converga muy pronto.

        # Se mide la "distancia" que existe entre la evaluacion
        # pasada y la actual.
        delta_eval = eval_solucion[0] - eval_act_consec
        abs_delta_eval = abs(delta_eval)
        dist_vals.append(abs_delta_eval)
        contador_conv += 1
        contador_salto = 0
        if contador_conv == disp_conv: # Si se cumple el criterio
                                       # del numero mediciones, se
                                       # calcula el promedio.
            radio_exploracion = np.sum(dist_vals)/len(dist_vals)
            print("Radio exploracion: ", radio_exploracion)

            # Si el radio de exploracion es menor o igual al valor
            # gatillo se genera el protocolo para reiniciar la
            # busqueda de manera aleatoria.
            if radio_exploracion <= valor_conv:
                # Definir el punto de inicio.
                mejor_n = None
                eval_solucion_costo = 99999
                eval_final = 0
                while mejor_n is None or not dentro_rango(mejor_n,
                                                          limites):
                    mejor_1 = np.random.uniform(limites[0],
```

```
                                                limites[1])
                    mejor_2 = np.random.uniform(limites[0],
                                                limites[1])
                    mejor_n = [mejor_1, mejor_2]

                    if dentro_rango(mejor_n, limites) == True:
                        # Se realiza la evaluacion de ese punto
                        # aleatorio.
                        eval_solucion_n, cuentas_n = bl_3_lineal(
    mejor_n, costo_total, costo_actual,

    costo_minimo, mejores_cuentas,

    num_particulas, simulador_local)
                        eval_solucion_costo = eval_solucion_n[0]

                        # Guardar la mejor solucion explorada hasta
                        # el momento.
                        if eval_solucion_n[0] < eval_final:
                            mejor_n = eval_solucion_n[1:]
                            mejor = mejor_n
                            eval_solucion = eval_solucion_n
                            cuentas = cuentas_n
                            eval_final = eval_solucion_n[0]

                        if eval_solucion_costo < mejor_eval[0]:
                            print("Salto: ", eval_solucion_n)
                            # El salto genera los nuevos valores
                            # para iniciar la siguiente iteracion.
                            mejor_n = eval_solucion_n[1:]
                            print("Mejor_n: ", mejor_n)
                            # En este caso se encontro una
                            # combinacion "mejor" que tiene un
                            # resultado mejor de VEE, entonces los
                            # valores de los parametros (mejor),
                            # la evaluacion de esta combinacion
                            # (mejor_eval) y las cuentas
                            # (mejores_cuentas) pasan
                            # a ser el mejor resultado actual.
                            mejor = mejor_n
                            eval_solucion = eval_solucion_n
                            cuentas = cuentas_n
                            break
                        else:
                            # Permite mantener la exploracion,
                            # generando valores por fuera de los
                            # limites que mantienen trabajando al
                            # "while loop".
                            mejor_1 = limites[1]+1
                            mejor_2 = limites[1]+1
                            mejor_n = [mejor_1, mejor_2]
                        if contador_salto == 1000:
                            print("No reinicio mejor, reinicio
    aleatorio")
```

```
227                             mejor_n = eval_solucion_n[1:]
228                             print("Eval: ", eval_solucion_n)
229                             # Se toma el inicio "mejor" como la
230                             # combinacion nueva de parametros
231                             # segun el salto, pero se considera
232                             # como solo una solucion que sera
233                             # comparada, en caso de no se mejor
234                             # solo se tomara el punto "mejor"
235                             # como nuestro nuevo inicio de
236                             # exploracion pero aun almacenando
237                             # la mejor solucion global
238                             mejor = mejor_n
239                             eval_solucion = eval_solucion_n
240                             cuentas = cuentas_n
241                             solucion = mejor_n
242                             contador_salto = 0
243                             eval_final = 0
244                             break
245 
246                         contador_salto += 1 # Contador en caso de
247                                             # no encontrar mejor
248                                             # salto.
249                     else:
250                         continue
251                 # Reiniciar contador de convergencia.
252                 contador_conv = 0
253                 dist_vals = []
254             else:
255                 contador_conv = 0
256                 dist_vals = []
257         #-------------------------------------------------------------#
258         # Evaluar para el "nuevo mejor" si existe.
259         if eval_solucion[0] < mejor_eval[0]:
260             mejor, mejor_eval = solucion, eval_solucion
261             mejores_cuentas = cuentas
262             print("Reinicio--> N: ", n, ", M_Ev: ", mejor_eval)
263             print("Cuentas: ", cuentas)
264             print("\n")
265             eval_act_consec = mejor_eval[0]
266     return mejor, mejor_eval, mejores_cuentas
267 #- Algoritmo de Busqueda Local Iterativa -#
268 
269 
270 #- Funcion objetivo o funcion de costo -#
271 def funcion_objetivo(x):
272     costo = 0
273     interaccion = 0
274     campo_mag = 0
275     # Calculo del costo de interaccion.
276     for i in range(len(x)-1):
277         if x[i] == x[i+1]:
278             interaccion += 0 # No suma conexion.
279         else:
280             interaccion -= 1 # Suma conexion, se usa signo (-)
```

```
                                # para mantener el estandar de VEE
                                # donde se evalua por un valor menor,
                                # podria cambiarse por singo (+)
                                # haciendo las modificaciones
                                # pertinentes.
    costo += interaccion
    return costo
#- Funcion objetivo o funcion de costo -#


############- Busqueda local MAX-CUT (caso lineal) VEE -#############
def bl_3_lineal(candidato, costo_total, costo_actual,
                costo_minimo, mejores_cuentas,
                num_particulas, simulador_local):

    # Valor de prueba para "beta" que corresponde al parametro
    # del primer operador mezclador.
    beta = candidato[1]

    # Valor de prueba para "gamma" que corresponde al parametro
    # del primer operador de fase.
    gamma = candidato[0]

    # Valor para almacenar el valor de energia esperado para
    # una iteracion.
    costo_aparicion = 0

    # Generacion de regirsto cuantico "q_reg" y registro
    # clasico "c_reg".
    q_reg = QuantumRegister(num_particulas, name="q_reg")
    c_reg = ClassicalRegister(num_particulas, name="c_reg")
    q_c = QuantumCircuit(q_reg,c_reg,name="q_c")

    #* Paso (1)
    #* El primer paso es generar la superposicion de estados
    #* iniciales, esto se logra aplicando compuertas Hadamard
    #* a los qubits del registro cuantico.
    for q in np.arange(num_particulas):
        q_c.h(q_reg[int(q)]) # Aplicamos la compuerta "h"
                            # (Hadamard) a los qubits.

    #* Paso (2)
    #* Se aplican las compuertas "Z" que corresponden al
    #* operador de fase, esto se realiza considerando las
    #* conexiones entre las particulas para el caso del
    #* Ising Spin Model y tambien se considera el valor de
    #* "gamma".
    for conexion in np.arange(num_particulas-1):
        q_c.cx(q_reg[conexion], q_reg[conexion+1])
        q_c.p(2*gamma, q_reg[conexion+1])
        q_c.cx(q_reg[conexion], q_reg[conexion+1])

    #** Se omite el "Paso (3)" del Ising Spin Model
    #** ya que no hay aporte de campo externo para
```

```
335     #** Max-Cut.
336 
337     #* Paso (3)
338     #* Aqui se agrega el operador mezclador, que corresponde
339     #* a los procesos de interferencia constructiva o
340     #* destructiva tomando en cuenta el valor de "beta".
341     for mez in np.arange(num_particulas):
342         q_c.rx(2*beta, mez)
343 
344     #* Paso (4)
345     #* Establecer las mediciones para el registro cuantico,
346     #* los valores de los estados base que se obtienen al
347     #* medir los qubits (registro cuantico) son almacenados
348     #* en los bits clasicos (registro clasico).
349     q_c.barrier()
350     q_c.measure(q_reg, c_reg)
351 
352     # Numero de experimentos que se correrar por cada par
353     # de valores gamma y beta.
354     n_experimentos = 1000
355 
356     # Ejecutar el circuito cuantico en el simulador.
357     trabajo = execute(q_c, simulador_local,
358                       shots= n_experimentos)
359 
360     # Extraer los resultados.
361     resultado = trabajo.result()
362 
363     # Extraer los datos estadisticos.
364     cuentas = resultado.get_counts(q_c)
365 
366     for elemento in cuentas.items():
367         costo = funcion_objetivo(elemento[0])
368         costo_ap = costo*float(elemento[1])
369         costo_aparicion += (costo_ap / n_experimentos)
370 
371     costo_actual = costo_aparicion
372     costo_minimo = [costo_actual, gamma, beta]
373     mejores_cuentas = cuentas
374 
375     return costo_minimo, mejores_cuentas
376 ############- Busqueda local MAX-CUT (caso lineal) VEE -#############
377 
378 
379 ###--------------------------- MAIN -----------------------------###
380 if __name__ == "__main__":
381 
382     #- Parametros para la busqueda local -#
383     # Limites de valores para "gamma" y "beta".
384     limites = [0, 2*m.pi]
385     # Total de iteraciones (para "ame").
386     n_iteraciones = 1000
387     # Tamano maximo de paso.
388     tam_paso = [-0.05, 0.05]
```

```
389     # Numero de reinicios.
390     n_reinicios = 100
391     # Tamano de perturbacion.
392     tam_per = [0.0, 1.0]
393     # Valores para el reinicio por convergencia.
394     valor_conv = 1.0 # Diferencia entre la respuesta evaluada
395                      # y la mejor respuesta.
396     disp_conv = 10 # Numero de ocurrencias de "valor_conv" para
397                    # que se pueda disparar el reinicio aleatorio.
398     #-------------------------------------#
399
400     #- Parametros para circuito cuantico -#
401     costo_total = 0
402     costo_actual = 0
403     costo_minimo = [0.0, 0.0, 0.0]
404     mejores_cuentas = []
405     num_particulas = 3
406     simulador_local = Aer.backends(name="qasm_simulator")[0]
407     #-------------------------------------#
408
409     mejor, mejor_eval, mejores_cuentas = busqueda_local_iterada_3p(
        disp_conv,
410                      valor_conv, limites, n_iteraciones,
411                      tam_paso, n_reinicios, tam_per, costo_total,
412                      costo_actual, costo_minimo, mejores_cuentas,
413                      num_particulas, simulador_local)
414
415     print("Mejor evaluacion: ", mejor_eval)
416     plot_histogram(mejores_cuentas)
417     plt.show()
418 ###--------------------------- MAIN -----------------------------###
```

Listing B.14: Código para simulación local usando BLI en el problema de Max-Cut de 3 nodos en configuración lineal.

El **Código B.14** es bastante similar al código usado para el problema de 3 partículas en configuración lineal, como se comentó anteriormente el funcionamiento de BLI realmente no muestra modificaciones considerables entre distintos problemas, esto es porque la estrategia de optimización solo se aplica a los parámetros que se usan en QAOA por lo que el único factor que afecta realmente a las funciones de BLI es el número de parámetros. Para el caso de la implementación de QAOA y su función de evaluación de costo se tienen modificaciones similares a las hechas para el método de BE en ISM, para el problema de Max-Cut se tiene que extraer la componente que incorpora el campo magnético externo y para la función de evaluación también se debe quitar la parte que contempla el costo de este campo para la evaluación del estado.

Código para problema de 4 nodos en configuración cíclica

Pasando al siguiente programa del problema de 4 nodos en configuración cíclica, solo se agregan las modificaciones realizadas a la etapa que corresponde a QAOA (únicas modificaciones importantes).

```
########--            Problema de 3 nodos Max-Cut           --#######
########--               usando Busqueda Local              --#######
#####################################################################


############################## CAMBIOS ******************************
# .
# .
# .
#- Funcion objetivo o funcion de costo -#
def funcion_objetivo(x, h):
    # .
    # .
    # .
    # Calculo del costo de interaccion.
    for i in range(len(x)):
        if x[i] == x[i-1]:
            interaccion += 0 # No suma conexion.
        else:
            interaccion -= 1 # Suma conexion, se usa signo (-)
                             # para mantener el estandar de VEE
                             # donde se evalua por un valor menor,
                             # podria cambiarse por singo (+)
                             # haciendo las modificaciones
                             # pertinentes.
    costo += interaccion
    # .
    # .
    # .

#################- Busqueda local (caso completo) -#################
    # .
    # .
    # .
    #* Paso (2)
    #* Se aplican las compuertas "Z" que corresponden al
    #* operador de fase, esto se realiza considerando las
    #* conexiones entre las particulas para el caso del
    #* Ising Spin Model y tambien se considera el valor de
    #* "gamma".
    for conexion in np.arange(num_particulas):
        q_c.cx(q_reg[conexion], q_reg[conexion-1])
        q_c.p(2*gamma, q_reg[conexion])
        q_c.cx(q_reg[conexion], q_reg[conexion-1])

    #** Se omite el "Paso (3)" del Ising Spin Model
    #** ya que no hay aporte de campo externo para
```

```
    #** Max-Cut.
    # .
    # .
    # .
#################- Busqueda local (caso completo) -#################


############################** CAMBIOS **############################
```

Listing B.15: Código para simulación local usando BLI en el problema de Max-Cut de 4 nodos en configuración cíclica.

De igual forma que para los códigos anteriores, vemos que el **Código B.15** solo modifica el modelo de QAOA y su función de evaluación, mientras que BLI no tiene modificaciones importantes para agregar.

Modificando el modelo de QAOA agregando dos parámetros para el operador mezclador se tiene el siguiente programa.

```
########--           Problema de 4 nodos Max-Cut          --#######
########--                 con dos BETAs                  --#######
########--              usando Busqueda Local             --#######
#####################################################################


############################** CAMBIOS **############################
# .
# .
# .
#- Funcion de verificacion -#
def dentro_rango(punto, limites):
    # Checar si un valor propuesto esta dentro del rango
    # permitido.
    if punto[0] >= limites[0] and punto[0] <= limites[1]:
        if punto[1] >= limites[0] and punto[1] <= limites[1]:
            if punto[2] >= limites[0] and punto[2] <= limites[1]:
                return True
            else:
                return False
        else:
            return False
    else:
        return False
#- Funcion de verificacion -#
    # .
    # .
    # .
#- Algoritmo de Ascenso en la Montana Estocastico -#
    # .
    # .
    # .
    while candidato is None or not dentro_rango(candidato,
```

```
                                                        limites):
            eleccion_a = np.random.choice([0,1,2])
            if eleccion_a == 0:
                candidato_1 = solucion[0] +\
                              np.random.uniform(limites[0],
                                                limites[1]) *\
                              np.random.uniform(tam_paso[0],
                                                tam_paso[1])
                candidato_2 = solucion[1]
                candidato_3 = solucion[2]
            elif eleccion_a == 1:
                candidato_1 = solucion[0]
                candidato_2 = solucion[1] +\
                              np.random.uniform(limites[0],
                                                limites[1]) *\
                              np.random.uniform(tam_paso[0],
                                                tam_paso[1])
                candidato_3 = solucion[2]
            else:
                candidato_1 = solucion[0]
                candidato_2 = solucion[1]
                candidato_3 = solucion[2] +\
                              np.random.uniform(limites[0],
                                                limites[1]) *\
                              np.random.uniform(tam_paso[0],
                                                tam_paso[1])

            candidato = [candidato_1, candidato_2, candidato_3]
        # .
        # .
        # .
#- Algoritmo de Ascenso en la Montana Estocastico -#

#- Algoritmo de Busqueda Local Iterativa -#
    # .
    # .
    # .
    # Definir el punto de inicio.
    mejor = None
    while mejor is None or not dentro_rango(mejor, limites):
        mejor_1 = np.random.uniform(limites[0], limites[1])
        mejor_2 = np.random.uniform(limites[0], limites[1])
        mejor_3 = np.random.uniform(limites[0], limites[1])
        mejor = [mejor_1, mejor_2, mejor_3]
    # .
    # .
    # .
        punto_inicio = None
        while punto_inicio is None or not dentro_rango(punto_inicio
    ,
                                                       limites):
            eleccion_b = np.random.choice([0,1,2])
            if eleccion_b == 0:
                punto_inicio_1 = mejor[0] +\
```

```
                        np.random.uniform(limites[0],
                                          limites[1]) *\
                        np.random.uniform(tam_per[0],
                                          tam_per[1])
            punto_inicio_2 = mejor[1]
            punto_inicio_3 = mejor[2]
        elif eleccion_b == 1:
            punto_inicio_1 = mejor[0]
            punto_inicio_2 = mejor[1] +\
                        np.random.uniform(limites[0],
                                          limites[1]) *\
                        np.random.uniform(tam_per[0],
                                          tam_per[1])
            punto_inicio_3 = mejor[2]
        else:
            punto_inicio_1 = mejor[0]
            punto_inicio_2 = mejor[1]
            punto_inicio_3 = mejor[2] +\
                        np.random.uniform(limites[0],
                                          limites[1]) *\
                        np.random.uniform(tam_per[0],
                                          tam_per[1])

        punto_inicio = [punto_inicio_1, punto_inicio_2,
punto_inicio_3]
        # .
        # .
        # .
        #--------------- REINICIOS POR CONVERGENCIA
---------------#
            # .
            # .
            # .
            while mejor_n is None or not dentro_rango(mejor_n,
                                                      limites):
                mejor_1 = np.random.uniform(limites[0], limites
[1])
                mejor_2 = np.random.uniform(limites[0], limites
[1])
                mejor_3 = np.random.uniform(limites[0], limites
[1])
                mejor_n = [mejor_1, mejor_2, mejor_3]
                # .
                # .
                # .
                    else:
                        # Permite mantener la exploracion,
                        # generando valores por fuera de los
                        # limites que mantienen trabajando al
                        # "while loop".
                        mejor_1 = limites[1]+1
                        mejor_2 = limites[1]+1
                        mejor_3 = limites[1]+1
                        mejor_n = [mejor_1, mejor_2, mejor_3]
```

```
                        # .
                        # .
                        # .
            #-------------- REINICIOS POR CONVERGENCIA
    ---------------#

#- Algoritmo de Busqueda Local Iterativa -#

############- Busqueda local MAX-CUT (caso ciclico) VEE -############
    # .
    # .
    # .
    # Valor de prueba para "beta_2" que corresponde al segundo
    # parametro del primer operador mezclador.
    beta_2 = candidato[2]

    # Valor de prueba para "beta" que corresponde al parametro
    # del primer operador mezclador.
    beta = candidato[1]

    # Valor de prueba para "gamma" que corresponde al parametro
    # del primer operador de fase.
    gamma = candidato[0]
    # .
    # .
    # .
    #* Paso (3.5) OPCIONAL
    #* Este paso se realiza para aumentar el numero
    #* de parametros dentro del operador mezclador,
    #* es importante agregar el nuevo parametro (ej.
    #* "beta_2") de manera entrelazada para garantizar
    #* que QAOA sea capaz de generar estados
    #* entrelazados y que esto a su vez aumente el
    #* espacio de busqueda posible del algoritmo.
    for ent in np.arange(num_particulas):
        q_c.cx(q_reg[ent], q_reg[ent-1])
    q_c.barrier()
    for mez2 in np.arange(num_particulas):
        q_c.rx(2*beta_2, mez2)
    q_c.barrier()
    # .
    # .
    # .
############- Busqueda local MAX-CUT (caso ciclico) VEE -############

############################*** CAMBIOS ***############################
```

Listing B.16: Código para simulación local usando BLI en el problema de Max-Cut de 4 nodos en configuración cíclica con dos parámetros β.

Al igual que en el caso de ISM vemos que el **Código B.16** que representa la modificación del modelo QAOA usando dos parámetros β es prácticamente idéntico, porque el aumento de un parámetro también incrementa en uno el número

de variables exploradas en BLI, y posteriormente esta nueva variable se agrega como tal al circuito cuántico generando la etapa de entrelazamiento de las dos βs (líneas 162 - 174).

Para el modelo de dos operadores de fase y dos operadores mezcladores (cuatro parámetros en total para QAOA) se usa un proceso parecido al anterior pero ahora agregando el nuevo parámetro extra, y también se modifica el circuito cuántico de QAOA como en el caso de ISM donde se usaba este tipo de modelo.

Código para problema de 5 nodos en configuración completa

Para el problema de 5 nodos en configuración completa solo se necesitaron los siguientes cambios para poder realizar la simulación correcta del circuito de esta instancia.

```
########--           Problema de 5 nodos Max-Cut          --#######
########--              usando Busqueda Local             --#######
#######################################################################


##############################*** CAMBIOS ***#############################

###########- Busqueda local MAX-CUT (caso completo) VEE -############
    # .
    # .
    # .
    #* Paso (2)
    #* Se aplican las compuertas "Z" que corresponden al
    #* operador de fase, esto se realiza considerando las
    #* conexiones entre las particulas para el caso del
    #* Ising Spin Model y tambien se considera el valor de
    #* "gamma".
    conexiones = []
    for conexion in np.arange(num_particulas):
        for conexion_2 in np.arange(num_particulas):
            if conexion != conexion_2:
                if [conexion, conexion_2] and [conexion_2, conexion
    ] not in conexiones:
                    q_c.cx(q_reg[conexion_2], q_reg[conexion])
                    q_c.p(2*gamma, q_reg[conexion_2])
                    q_c.cx(q_reg[conexion_2], q_reg[conexion])
                    conexiones.append([conexion, conexion_2])
                    conexiones.append([conexion_2, conexion])
                else:
                    continue
            else:
                continue
    q_c.barrier()
    # .
    # .
    # .
```

```
##########- Busqueda local MAX-CUT (caso completo) VEE -###########

############################*** CAMBIOS ***############################
```

Listing B.17: Código para simulación local usando BLI en el problema de Max-Cut de 5 nodos en configuración completa.

Como se observa en el **Código B.17** la única modificación importante que se presenta es respecto al circuito cuántico de QAOA donde se deben incorporar las conexiones entre todos los nodos del problema.

Para los modelos modificados de 2 parámetros β y cuatro parámetros (2 γ y 2 β) se siguieron pasos similares a los programas anteriores, en presencia de más parámetros se va incrementando el número de variables a explorar por parte de BLI y a su vez se incorporan a QAOA para interactuar en las etapas correspondientes y con las compuertas cuánticas necesarias.

B.2. Programas para simulaciones reales

Los programas para las simulaciones reales son muy parecidos a los programas generados para las simulaciones locales, dado que el método de optimización tanto para BE como BLI no se puede correr en las computadoras de IBM Q (debido al tiempo permitido de procesamiento) todos los programas de las simulaciones reales solo comprenden a la etapa de los circuitos cuánticos dentro de QAOA. Por lo anterior mencionado, no es necesario agregar el código de estos programas porque son prácticamente los mismos de las simulaciones locales (sin el método de optimización) y considerando el cambio del simulador **qasm_simulator** por alguno de los dispositivos cuánticos: **ibmq_santiago**, **ibmq_bogota** o **ibmq_manila**.

Apéndice C

Simulaciones Locales

En este apéndice llamado **Simulaciones Locales** se anexan los resultados complementarios a las simulaciones locales, estos resultados son relevantes para su estudio, pero no forman parte del cuerpo principal del trabajo, el cual se busca mantener simple y conciso para resaltar el aporte de este trabajo.

C.1. Simulaciones complementarias para ISM usando BE

Esta sección contiene las simulaciones complementarias generadas para los problemas de ISM usando BE, estas simulaciones contienen diferentes elementos que fueron variados como el número de divisiones, el número de parámetros por operador, etc.

C.1.1. Simulaciones complementarias de ISM usando BE para configuración lineal de 3 partículas

Para terminar con estos primeros resultados del problema de 3 partículas en configuración lineal, ahora se presentan los datos generados para 100 divisiones del rango de valores de los parámetros.

Como se observa en la **Tabla C.1** vemos que el aumentar el número de divisiones para el rango de valores de γ y β ayudo a tener un mayor número de

Estado obtenido	$\|001\rangle$
Número de apariciones	792
CTA	-2772.0
Valor de γ	1.382301 π
Valor de β	1.570796 π

Tabla C.1: Resultados de problema ISM con configuración lineal de 3 partículas con 100 divisiones.

apariciones, y un valor más alto de CTA; también se tiene un resultado bastante interesante y es que el valor de γ entre las 50 y 100 divisiones no tuvo una diferencia considerable, mientras que el valor de β si tuvo un cambio de casi 2 décimas. Este es mejor resultado de los tres experimentos de diferentes divisiones permitidas para ejecutar la Búsqueda Exhaustiva, sin embargo, también fue la más tardada de todas, ya que para las 25 y 50 divisiones el resultado del programa y la gráfica fueron generadas en menos de 3 minutos, pero para el de 100 el tiempo de ejecución se extendió alrededor de los 15 minutos, esto presenta un dilema para lo que se busca como solución del problema, ya que si bien los resultados anteriores no fueron mejores que el de 100 divisiones, se puede decir que son lo "suficientemente" buenos si lo que se busca es rapidez en la respuesta.

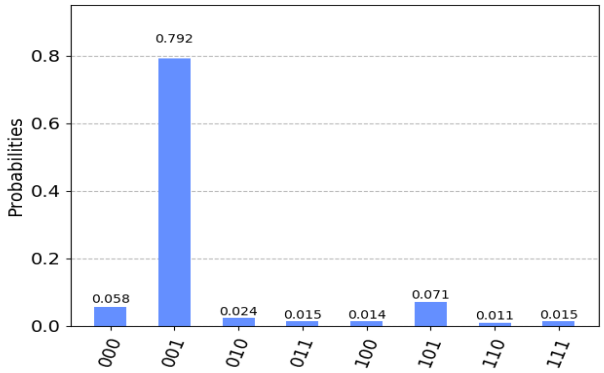


Figura C.1: Histograma de mejor combinación de parámetros $\gamma = 1.382301\pi$ y $\beta = 1.570796\pi$ del problema de 3 partículas en configuración lineal.

En la **Figura C.1** vemos que la probabilidad de medición del estado óptimo $|001\rangle$ es casi del 80 % lo cual es una respuesta bastante aceptable, además, se puede ver que el aumentar el número de divisiones mejora la respuesta de QAOA usando BE, pero aparentemente llegando a un límite de precisión de la respuesta, esto es algo de esperarse, ya que el tipo de QAOA implementado presenta una baja profundidad, (poco número de compuertas cuánticas, y solo 2 hiperparámetros para optimizar) se podría optar por implementar una mayor profundidad, es decir, agregar más etapas de operadores de fase y mezcladores (aumentar el número de hiperparámetros a optimizar) para poder mejorar aún más la respuesta de QAOA pero a costa de un mayor costo computacional, tanto cuántico como clásico.

Estado obtenido	$\|1101\rangle$
Número de apariciones	536
CTA	-3162.4
Valor de γ	4.667509 π
Valor de β_1	1.615676 π
Valor de β_2	0.0 π

Tabla C.2: Resultados de problema ISM con configuración cíclica de 4 partículas con 35 divisiones y dos parámetros β_1 y β_2.

C.1.2. Simulaciones complementarias usando BE para configuración cíclica de 4 partículas

Continuando con las simulaciones para el problema de 4 partículas en el caso del aumento a dos parámetros β_1 y β_2 con 35 divisiones se obtuvieron los siguientes resultados usando el método de evaluación con estado óptimo y CTA.

Y el histograma generado para la mejor combinación de parámetros γ, β_1 y β_2.

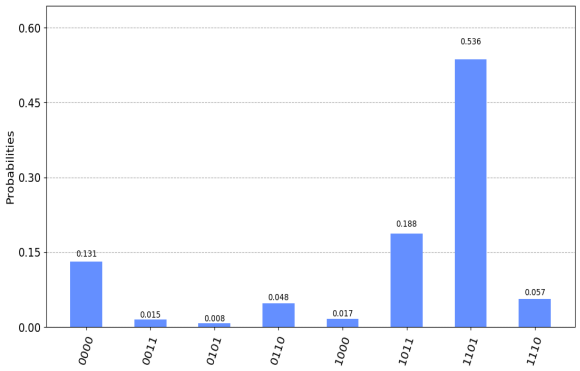


Figura C.2: Histograma de mejor combinación de parámetros $\gamma = 4.667509\pi$, $\beta_1 = 1.615676\pi$ y $\beta_2 = 0.0\pi$ (dos parámetros mezcladores) del problema de 4 partículas en configuración cíclica.

C.1.3. Simulaciones complementarias usando BE para configuración completa de 5 partículas

Para la simulación con 50 divisiones con método de evaluación por búsqueda de estado óptimo y CTA, los resultados generados son los siguientes.

Los resultados de la tabla muestran que no existe una mejora sustancial al aumentar el número de divisiones para aumentar la probabilidad de aparición del

Estado obtenido	$\|00000\rangle$
Número de apariciones	402
CTA	-4381.8
Valor de γ	5.026548 π
Valor de β	4.398229 π

Tabla C.3: Resultados de problema ISM con configuración completa de 5 partículas con 50 divisiones.

Estado obtenido	$\|00000\rangle$
VEE	-7.9822
Valor de γ	5.026548 π
Valor de β	4.775220 π

Tabla C.4: Resultados de problema ISM con configuración completa de 5 partículas con 50 divisiones usando VEE.

estado óptimo del sistema.

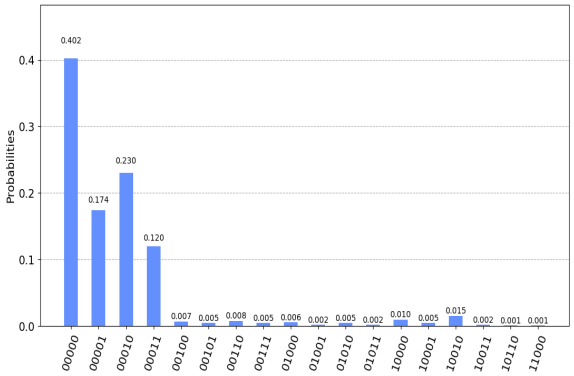


Figura C.3: Histograma de mejor combinación de parámetros $\gamma = 5.026548\pi$ y $\beta = 4.398229\pi$ del problema de 5 partículas en configuración completa.

De igual forma, el histograma de la **Figura C.3** vemos que no existe mucha diferencia al compararlo con el caso de 25 divisiones.

Para el caso del uso de VEE como método de evaluación para la simulación se obtuvieron los siguientes datos.

De igual forma que en el otro método de evaluación, se observa que la diferencia en el valor de VEE logrado con 50 divisiones es muy similar al de 25 divisiones. Existe también una diferencia entre los dos métodos de evaluación es que los valores de β son diferentes por unos decimales.

También, el histograma generado de la **Figura C.4** muestra resultados si-

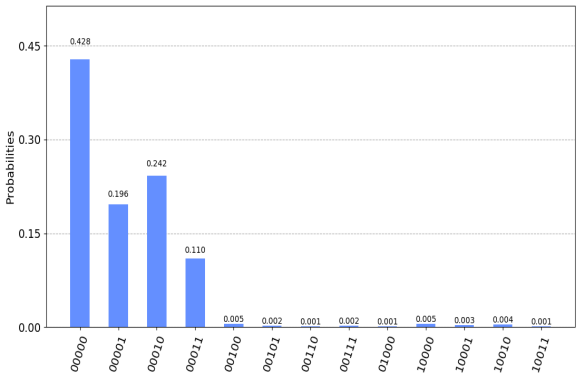


Figura C.4: Histograma de mejor combinación de parámetros $\gamma = 5.026548\pi$ y $\beta = 4.775220\pi$ del problema de 5 partículas en configuración completa usando VEE.

milares a las otras simulaciones, por ello como se comentó que no era necesario agregar estos resultados al cuerpo principal de la tesis, sin embargo, era importante plasmar los resultados porque aumentar o disminuir la precisión de los valores posibles para los parámetros γ y β puede ser un factor importante para aumentar la probabilidad de aparición del estado óptimo.

C.2. Simulaciones complementarias para Max-Cut usando BE

Esta sección contiene las simulaciones complementarias generadas para los problemas de Max-Cut usando BE, estas simulaciones no se consideran tan relevantes para incluirlas en el cuerpo principal del trabajo, sin embargo, presentan evidencia de los análisis y conclusiones realizadas. Todas las simulaciones se generaron usando VEE como evaluación.

C.2.1. Simulaciones complementarias de Max-Cut usando BE para configuración lineal de 3 partículas

Para esta simulación se aumentó a 50 el número de divisiones posibles para el rango de parámetros.

Como se puede comprobar en la **Tabla C.5** vemos que los resultados son bastante similares a los de 25 divisiones.

Estado obtenido	$\|010\rangle$
VEE	1.673
Valor de γ	3.644247 π
Valor de β	2.261946 π

Tabla C.5: Resultados de problema Max-Cut con configuración lineal de 3 partículas con 50 divisiones usando VEE.

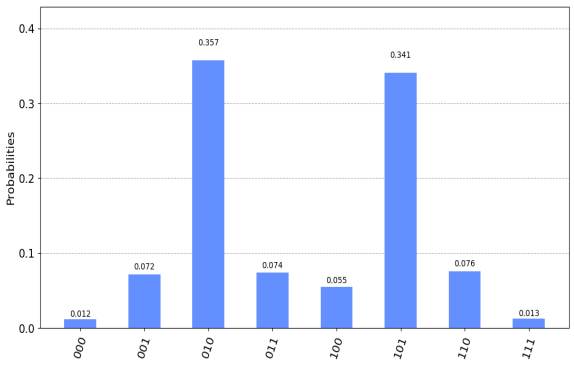


Figura C.5: Histograma de mejor combinación de parámetros $\gamma = 3.644247\pi$ y $\beta = 2.261946\pi$ del problema de 3 partículas en configuración lineal.